

CORE DISCUSSION PAPER

2012/02

Subgradient methods for huge-scale optimization problems

Yu. Nesterov *

January, 2012

Abstract

We consider a new class of huge-scale problems, the problems with *sparse subgradients*. The most important functions of this type are piece-wise linear. For optimization problems with uniform sparsity of corresponding linear operators, we suggest a very efficient implementation of subgradient iterations, which total cost depends *logarithmically* in the dimension. This technique is based on a recursive update of the results of matrix/vector products and the values of symmetric functions. It works well, for example, for matrices with few nonzero diagonals and for max-type functions.

We show that the updating technique can be efficiently coupled with the simplest subgradient methods, the unconstrained minimization method by B.Polyak, and the constrained minimization scheme by N.Shor. Similar results can be obtained for a new non-smooth random variant of a coordinate descent scheme. We present also the promising results of preliminary computational experiments.

Keywords: Nonsmooth convex optimization, complexity bounds, subgradient methods, huge-scale problems.

*Center for Operations Research and Econometrics (CORE), Catholic University of Louvain (UCL), 34 voie du Roman Pays, 1348 Louvain-la-Neuve, Belgium; e-mail: nesterov@core.ucl.ac.be.

The research presented in this paper was partially supported by the Laboratory of Structural Methods of Data Analysis in Predictive Modeling, MIPT, through the RF government grant, ag.11.G34.31.0073

1 Introduction

Motivation. In Convex Optimization, the size of the problem plays a crucial role for the choice of minimization scheme. For problem of small size (see Table 1), all auxiliary operations are feasible, and we can apply the most sophisticated schemes (e.g. Inscribed Ellipsoid Method [1]). Medium-size problems should be treated by polynomial-time interior-point methods [6], which require matrix inversions. Large-scale problems can be solved by the fast gradient schemes [3], or by primal-dual subgradient methods (e.g. [4]). In the later class, the matrix-vector multiplication is still feasible. However, we can work only with the sparse matrices.

Class	Operations	Dimension	Iteration Cost		Memory units	
			low	high		
Small-size	All	$10^0 - 10^2$	n^3	n^4	Kilobyte:	10^3
Medium-size	A^{-1}	$10^3 - 10^4$	n^2	n^3	Megabyte:	10^6
Large-scale	Ax	$10^5 - 10^7$	n	n^2	Gigabyte:	10^9
Huge-scale	$x + y$	$10^8 - 10^{12}$	$\log n$	n	Terabyte:	10^{12}

Table 1. Problem sizes, operations and memory.

In the last years we can observe an increasing interest to the huge-scale problems. These problems are so big, that even the simplest vector operation require considerable computational efforts. Corresponding matrices are very sparse and have typically a constant number of nonzero elements in each row. This type of structure arise frequently in image processing, finite-elements models, problems related to internet, telecommunications and partial differential equations.

At the current moment, there are some promising results on adopting the smoothing technique to the special equilibrium problems of very big size [7]. However, the main methods for solving the huge-scale problems remain the Coordinate Descent Schemes (e.g. [2]). Recently, the random variants of these schemes were endowed with the global complexity bounds [5, 9] for the classes of convex problems with Lipschitz continuous gradient. In these methods, each iteration consists in updating a single entry of the current test point. Therefore, it is very cheap. However, the total number of iterations, which is necessary for obtaining an approximate solution of the problem, is typically higher than the corresponding number of iterations of the full-gradient scheme.

In this paper, we consider a new class of huge-scale problems, the problems with *sparse subgradients*. For smooth functions this is a very rare feature. Indeed, for quadratic function $f(y) = \frac{1}{2}\langle Ay, y \rangle$ its gradient $\nabla f(y) = Ay$ usually is not sparse even for a sparse matrix A . For nonsmooth functions, the situation is different. Indeed, the subgradients of function $f(y) = \max_{1 \leq i \leq m} \langle a_i, y \rangle$ are sparse provided that all vectors a_i share this property.

The results of this paper are based on the following simple observation. Consider the function $f(y) = \max_{1 \leq i \leq m} \langle a_i, y \rangle$ with sparse matrix $A = (a_1, \dots, a_m)$. Let $f'(y) = a_{i(y)}$ be its subgradient at point $y \in R^n$. Then the subgradient step

$$y_+ = y - hf'(y) \tag{1.1}$$

changes only a few entries of vector y . This means that the vector $u_+ = A^T y_+$ differs from $u = A^T y$ also in a few positions only. Thus, the function value $f(y_+)$ can be easily updated

provided that we have an efficient procedure for recomputing the maximum of m values. It can be shown that for matrix A with uniform filling, such *full-gradient* iteration can be performed in $O(\gamma^2(A) \cdot mn \cdot \log_2 m)$ operations, where $\gamma(A)$ is the sparsity coefficient of matrix A . Recall that the usual matrix-vector product $A^T y$, which is necessary for computing the value $f(y)$, needs $\gamma(A) \cdot mn$ operations. Note that in huge-scale problems the sparsity coefficient is usually very small.

Another important example of a sparse matrix is the matrix with limited number of nonzero diagonals. If this number does not exceed q , then the complexity of each iteration of scheme (1.1) does not exceed $O(q^2 \ln m)$ operations. Thus, it does not depend on the dimension of the vector of variables and grows very slowly with the number of linear pieces.

Not too many subgradient methods can be efficiently coupled with the above updating technique. Indeed, most of them need at least $O(n)$ operations per iteration (e.g. [4]). We managed to find only two simple subgradient methods for nonsmooth minimization, which can get a full advantage from the sparsity of subgradients. These are the oldest subgradient methods by Polyak [8] and Shor [10]. In both schemes, despite to a remarkable drop in the complexity of each iteration, we preserve the initial estimate for the number of iterations $O(\frac{1}{\epsilon^2})$, where ϵ is the desired accuracy in the function value. Similar results can be obtained for a new random coordinate descent scheme, which takes into account only nonzero components of subgradients.

Contents. In Section 2 we analyze the complexity of the update of the matrix-vector product in the case when the variation of the vector is sparse. In Section 3 we describe a simple technique for updating the value of a symmetric function of n variables when only one entry is changing. Its complexity is $O(\log_2 n)$ operations. In Section 4 we give the complexity bounds and the stepsize rules for two subgradient methods, the unconstrained minimization scheme by Polyak [8], and a constrained minimization method by Shor [10]. In Section 5 we consider huge-scale problems of the special sparse block structure, and discuss complexity of sparse update of matrix-vector products. In Section 6 we derive the complexity bounds for the random coordinate descent method for minimizing nonsmooth convex functions with sparse subgradients. In Section 7 we give the results of preliminary computational experiments. Finally, in Section 8 we discuss possible extension of the results onto the matrices with nonuniform filling.

Notation. For a column vector $x = (x^{(1)}, \dots, x^{(k)})^T \in R^k$ we denote by $p(x) \leq k$ the number of nonzero elements in its coordinate representation. The set of nonzero elements of x is denoted by $\sigma(x) \subseteq \{1, \dots, k\}$. If the sparsity is introduced with respect to a block structure, we use notation $\sigma_b(x)$.

We often use the *sparsity coefficient* of vector x ;

$$\gamma(x) \stackrel{\text{def}}{=} \frac{p(x)}{\dim x} \leq 1, \quad (1.2)$$

where $\dim x \equiv k$ is the dimension of the corresponding object. The same notation is used also for square matrices.

The above notation is naturally extended onto the block structure. In this case, $p_b(x)$ denotes the number of nonzero blocks in x (it can be a vector or a matrix), and $\gamma_b(x) = \frac{p_b(x)}{\dim_b x}$, where $\dim_b(x)$ is the number of nonzero blocks in the corresponding object. Finally the set $\sigma_b(x)$ contains the indices of nonzero blocks in x .

We denote by $e_j \in R^n$ the j th coordinate vector in R^n . The same notation is used for different dimensions. However, its exact sense is always clear from the context.

Finally, $\mathcal{E}(\xi)$ denotes the expectation of random variable ξ , and $\mathcal{E}(\xi_1|\xi_2)$ the conditional expectation of ξ_1 given the event ξ_2 . In all situations, our random variables are discrete.

2 Updating the sparse matrix-vector product

In many numerical schemes, the cost of internal matrix-vector multiplications is responsible for the leading term in the total complexity of the method. For a general dense matrix $A \in R^{M \times N}$, and arbitrary vector $x \in R^N$, computation of the product Ax usually needs MN arithmetic operations¹. The situation is changing if the matrix A is sparse. Assume that the matrix A is stored as a list of nonzero elements. Then the product Ax can be computed in $p(A)$ operations. Note that

$$p(A) \stackrel{(1.2)}{=} \gamma(A) \cdot MN. \quad (2.1)$$

Thus, the initial complexity is reduced by a factor of $\gamma(A)$.

For sparse matrices of high dimension, we usually have $\gamma(A) \ll 1$. In the sequel, we consider only matrices with nonzero rows and columns. Then, $p(A) \geq \max\{M, N\}$. Therefore,

$$\gamma(A) \geq \max\left\{\frac{1}{M}, \frac{1}{N}\right\}. \quad (2.2)$$

Let us show that the complexity of a *sparse* matrix-vector multiplication can be significantly reduced by a *recursive update* of the product. Indeed, let us assume that the vector $y = Ax$ is already computed. We need to compute a new vector $y_+ = Ax_+$, where

$$x_+ = x + d, \quad (2.3)$$

and the vector d is sparse. Then

$$y_+ = y + \sum_{j \in \sigma(d)} d^{(j)} \cdot Ae_j. \quad (2.4)$$

The complexity of this update is equal to

$$\kappa_A(d) \stackrel{\text{def}}{=} \sum_{j \in \sigma(d)} p(Ae_j). \quad (2.5)$$

As compared with the cost of the direct dense multiplication MN , this value can be very small.

Lemma 1

$$\kappa_A(d) = \gamma(d) \cdot \frac{1}{p(d)} \sum_{j \in \sigma(d)} \gamma(Ae_j) \cdot MN \leq \gamma(d) \cdot \max_{j \in \sigma(d)} \gamma(Ae_j) \cdot MN. \quad (2.6)$$

¹We count as one operation the pair of operations formed by real multiplication and addition.

Proof:

Since $Ae_j \in R^M$ and $d \in R^N$, we have

$$\begin{aligned}\kappa_A(d) &\stackrel{(1.2)}{=} M \sum_{j \in \sigma(d)} \gamma(Ae_j) = MN \cdot \frac{p(d)}{N} \cdot \frac{1}{p(d)} \sum_{j \in \sigma(d)} \gamma(Ae_j) \\ &\stackrel{(1.2)}{=} \gamma(d) \cdot \frac{1}{p(d)} \sum_{j \in \sigma(d)} \gamma(Ae_j) \cdot MN.\end{aligned}$$

□

Corollary 1 *Assume that the matrix $A \in R^{M \times N}$ has a uniform filling:*

$$\gamma(Ae_j) \leq c_u \cdot \gamma(A), \quad j = 1, \dots, N,$$

where $c_u \geq 1$ is an absolute constant. Assume also that $\gamma(d) \leq c_u \cdot \gamma(A)$. Then

$$\kappa_A(d) \leq c_u^2 \gamma^2(A) \cdot MN. \quad (2.7)$$

Comparing this estimate with (2.1), we can see that the sparse updating strategy (2.4) is in $O(\gamma(A))$ times cheaper than the direct sparse matrix-vector multiplication.

Very often, vector d in the sparse update (2.3) is proportional to one of the row of matrix A . In this case, the level of sparsity of matrix A can be well described by the following characteristic:

$$\kappa(A) \stackrel{\text{def}}{=} \max_{1 \leq i \leq M} \kappa_A(A^T e_i) = \max_{1 \leq i \leq M} \sum_{j \in \sigma(A^T e_i)} p(Ae_j), \quad (2.8)$$

which we call the *row capacity* of matrix A . In many practical applications (e.g. matrices with few nonzero diagonals), the row capacity is constant and does not depend on the matrix size.

3 Fast updates in short computational trees

Let $f(x)$ be a function of n variables. We call this function short-tree representable, if its value can be computed by a short binary tree with the height being proportional to $\ln n$.

In order to illustrate the main idea, let us assume that $n = 2^k$ for some $k \geq 1$, and the computational tree has $k + 1$ levels. At the level zero we have n cells containing the entries of vector x :

$$v_{0,i} = x^{(i)}, \quad i = 1, \dots, n$$

The size of each next level is a half of the size of the previous one. The values at the next level are computed by the recursion:

$$v_{i+1,j} = \psi_{i+1,j}(v_{i,2j-1}, v_{i,2j}), \quad j = 1, \dots, 2^{k-i-1}, \quad i = 0, \dots, k-1, \quad (3.1)$$

where $\psi_{i,j}$ are some functions of two variables (see Figure 1).

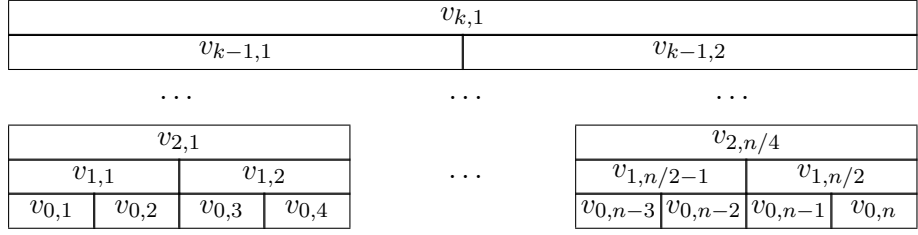


Figure 1. Short computational binary tree.

Let us present some important examples of such functions. All of them are symmetric functions of n variables, generated by symmetric functions of two variables.

$$\begin{aligned}
 f(x) &= \|x\|_p, \quad p \geq 1, & \psi_{i,j}(t_1, t_2) &\equiv [|t_1|^p + |t_2|^p]^{1/p}, \\
 f(x) &= \ln \left(\sum_{i=1}^n e^{x^{(i)}} \right), & \psi_{i,j}(t_1, t_2) &\equiv \ln(e^{t_1} + e^{t_2}), \\
 f(x) &= \max \{x^{(1)}, \dots, x^{(n)}\}, & \psi_{i,j}(t_1, t_2) &\equiv \max \{t_1, t_2\}.
 \end{aligned} \tag{3.2}$$

For all these functions, the computational binary tree has the minimal height.

Short-tree computation of the value of such a function has the following advantages.

- This scheme requires only $n - 1$ auxiliary cells, provided that the vector of variables x is stored at the zero level of the tree.
- Computation of its value needs $n - 1$ applications of functions $\psi_{i,j}(\cdot, \cdot)$.
- The computation of value $f(x_+)$ needs only $k \equiv \log_2 n$ applications of functions $\psi_{i,j}(\cdot, \cdot)$, provided that $f(x)$ is already computed, and x_+ differs from x in one entry only. Indeed, for such an *update*, we need to modify only the cells which are strictly above the new entry, one for each level (see Figure 1). Note that this updating strategy does not cause any error accumulation.

We will see later that the last feature opens a possibility for development of the pure subgradient minimization schemes with *sublinear iteration cost*.

4 Simple subgradient methods

In this section we consider two simple subgradient methods for solving nonsmooth optimization problems. At each iteration, both methods allow a partial update of the test points provided that the current subgradient is sparse and the feasible set is separable. Note that usually, the sparsity of the gradient is a feature of nonsmooth functions.

Let us start from the following simple problem:

$$\min_{x \in Q} f(x), \tag{4.1}$$

where Q is a closed convex set in R^n , and f is a closed convex function. Our main assumptions are as follows.

1. At each point $x \in Q$ we are able to compute a subgradient $f'(x)$ of function f .
2. These subgradients are uniformly bounded on Q :

$$\|f'(x)\| \leq L(f), \quad x \in Q. \quad (4.2)$$

3. The set Q is simple. This means that for any $x \in R^n$ we can easily compute its Eucleden projection $\pi_Q(x)$ onto the set Q .
4. The problem (4.1) is feasible, and its optimal set X_* is nonempty.
5. The optimal value f^* of problem (4.1) is known.

Consider the following optimization scheme [8]:

$$x_0 \in Q, \quad x_{k+1} = \pi_Q \left(x_k - \frac{f(x_k) - f^*}{\|f'(x_k)\|^2} f'(x_k) \right), \quad k \geq 0. \quad (4.3)$$

For the reader convenience, we present its convergence results with a complete proof. Denote

$$f_k^* = \min_{0 \leq i \leq k} f(x_i), \quad L_k(f) = \max_{0 \leq i \leq k} \|f'(x_i)\|.$$

Theorem 1 *For any $k \geq 0$ we have:*

$$f_k^* - f^* \leq \frac{L_k(f) \|x_0 - \pi_{X_*}(x_0)\|}{(k+1)^{1/2}}. \quad (4.4)$$

Moreover, for any $k \geq 0$ we have

$$\|x_k - x^*\| \leq \|x_0 - x^*\|, \quad \forall x^* \in X_*. \quad (4.5)$$

Hence, the sequence $\{x_k\}$ converges to a single point $\bar{x}^* \in X_*$.

Proof:

Let us fix an arbitrary $x^* \in X_*$. Denote $r_k(x^*) = \|x_k - x^*\|$. Then,

$$\begin{aligned} r_{k+1}^2(x^*) &\leq \left\| x_k - x^* - \frac{f(x_k) - f^*}{\|f'(x_k)\|^2} f'(x_k) \right\|^2 \\ &= r_k^2(x^*) - 2 \frac{f(x_k) - f^*}{\|f'(x_k)\|^2} \langle f'(x_k), x_k - x^* \rangle + \frac{(f(x_k) - f^*)^2}{\|f'(x_k)\|^2} \\ &\leq r_k^2(x^*) - \frac{(f(x_k) - f^*)^2}{\|f'(x_k)\|^2} \leq r_k^2(x^*) - \frac{(f_k^* - f^*)^2}{L_k^2(f)}. \end{aligned}$$

This proves inequality (4.4). On the other hand, from the latter reasoning, we have

$$\|x_{k+1} - x^*\|^2 \leq \|x_k - x^*\|^2 \quad \forall x^* \in X_*.$$

Hence, (4.5) is valid, and the sequence $\{x_k\}$ has only one limit point $\bar{x}^* \in X_*$. $\square$

Corollary 2 *Assume that X_* has a recession direction d_* . Then for any $k \geq 0$ we have*

$$\begin{aligned} \|x_k - \pi_{X_*}(x_0)\| &\leq \|x_0 - \pi_{X_*}(x_0)\|, \\ \langle d_*, x_k \rangle &\geq \langle d_*, x_0 \rangle. \end{aligned} \quad (4.6)$$

Proof:

Note that $x^*(\alpha) \stackrel{\text{def}}{=} \pi_{X_*}(x_0) + \alpha d_* \in X_*$ for any $\alpha \geq 0$. Therefore,

$$\begin{aligned} \|x_k - x^*(\alpha)\|^2 &= \|x_k - \pi_{X_*}(x_0)\|^2 - 2\alpha \langle d_*, x_k - \pi_{X_*}(x_0) \rangle + \alpha^2 \|d_*\|^2 \\ (4.5) \quad &\leq \|x_0 - x^*(\alpha)\|^2 = \|x_0 - \pi_{X_*}(x_0)\|^2 - 2\alpha \langle d_*, x_0 - \pi_{X_*}(x_0) \rangle + \alpha^2 \|d_*\|^2. \end{aligned}$$

Taking in this inequality $\alpha = 0$ and $\alpha \rightarrow +\infty$, we get both inequalities in (4.6), $\square$

Consider now an optimization problem with single functional constraint:

$$\min_{x \in Q} \{f(x) : g(x) \leq 0\}, \quad (4.7)$$

where Q is a closed convex set in R^n , and f and g are closed convex functions, which subgradients are uniformly bounded on Q . For this problem, we keep the first four assumptions introduced for problem (4.1). Since its feasible set is nonempty, we have

$$g'(x) \neq 0 \quad (4.8)$$

for all $x \in Q$ with $g(x) > 0$.

Consider the following method. It has two parameters: the step-size length $h > 0$, and the iteration limit N .

Method $SG_N(h)$.

For $k = 0, \dots, N - 1$ iterate:

If $g(x_k) > h \|g'(x_k)\|$, then (A): $x_{k+1} = \pi_Q \left(x_k - \frac{g(x_k)}{\|g'(x_k)\|^2} g'(x_k) \right)$,

else (B): $x_{k+1} = \pi_Q \left(x_k - \frac{h}{\|f'(x_k)\|} f'(x_k) \right)$.

(4.9)

For this method, denote by $\mathcal{F}_k \subseteq \{0, \dots, k\}$ the set of numbers of iterations of type (B) with cardinality k_f , and $k_g \stackrel{\text{def}}{=} k - k_f$. Denote

$$\begin{aligned} f_k^* &= \min_{i \in \mathcal{F}_k} f(x_i), \quad L_k(f) = \max_{i \in \mathcal{F}_k} \|f'(x_i)\|, \\ g_k^* &= \max_{i \in \mathcal{F}_k} g(x_i), \quad L_k(g) = \max_{i \notin \mathcal{F}_k} \|g'(x_i)\|. \end{aligned}$$

Let us fix an arbitrary point x feasible to problem (4.7). Denote $r_k(x) \stackrel{\text{def}}{=} \|x_k - x\|$.

Theorem 2 *If $N > r_0^2(x)/h^2$, then $\mathcal{F}_N \neq \emptyset$ and*

$$f_N^* - f(x) \leq hL_N(f), \quad g_N^* \leq hL_N(g). \quad (4.10)$$

Proof:

If $k \notin \mathcal{F}_N$, then

$$\begin{aligned} r_{k+1}^2(x) &\leq r_k^2(x) - 2 \frac{g(x_k)}{\|g'(x_k)\|^2} \langle g'(x_k), x_k - x \rangle + \frac{g^2(x_k)}{\|g'(x_k)\|^2} \\ &\leq r_k^2(x) - \frac{g^2(x_k)}{\|g'(x_k)\|^2} \leq r_k^2(x) - h^2. \end{aligned} \quad (4.11)$$

Hence, for $k > r_0^2(x)/h^2$ we have $\mathcal{F}_k \neq \emptyset$. Further, if $k \in \mathcal{F}_N$, then

$$\begin{aligned} r_{k+1}^2(x) &\leq r_k^2(x) - \frac{2h}{\|f'(x_k)\|} \langle f'(x_k), x_k - x \rangle + h^2 \\ &\leq r_k^2(x) - \frac{2h}{L_N(f)} (f(x_k) - f(x)) + h^2 \\ &\leq r_k^2(x) - \frac{2h}{L_N(f)} (f_N^* - f(x)) + h^2. \end{aligned} \quad (4.12)$$

Summing up these inequalities, we obtain

$$\begin{aligned} f_N^* - f(x) &\leq \frac{L_N(f)}{2hN_f} [r_0^2(x) + (N_f - N_g)h^2] \\ &= hL_N(f) + \frac{L_N(f)}{2hN_f} [r_0^2(x) - Nh^2] \leq hL_N(f). \end{aligned} \quad (4.13)$$

It remains to note that for all $k \in \mathcal{F}_N$ we have $g(x_k) \leq h\|g'(x_k)\| \leq hL_N(g)$. $\square$

Let us discuss now several strategies for applying method (4.9) to the problem (4.7). Our goal is to find an ϵ -solution $\bar{x} \in Q$ of this problem:

$$f(\bar{x}) - f(x^*) \leq \epsilon, \quad g(\bar{x}) \leq \epsilon. \quad (4.14)$$

1. Lipschitz constants $L(f)$, $L(g)$ are known. This is true, for example, for Linear Programming problems. Then, we can take

$$h = \frac{\epsilon}{\max\{L(f), L(g)\}}. \quad (4.15)$$

If the estimate $R_0 \geq \|x_0 - x^*\|$ is known, then we can use method (4.9) with h as in (4.15), and

$$N = \left\lfloor \frac{1}{h^2} R_0^2 \right\rfloor + 1. \quad (4.16)$$

Consider the case when R_0 is not known. Denote

$$Q(r) = \{x \in Q : \|x - x_0\| \leq r\},$$

$$x^*(r) \in \text{Arg min}_{x \in Q(r)} \{f(x) : g(x) \leq 0\}.$$

There are two ways to proceed.

- We choose N as the main control parameter. Define $r_N = \frac{1}{h} N^{1/2}$. Then the output of the method (4.9) can be interpreted as follows:

a. If $\mathcal{F}_N = \emptyset$, then there is no feasible points inside the set $Q(r_N)$.

b. If $\mathcal{F}_N \neq \emptyset$, then $f_N^* - f(x^*(r_N)) \leq \epsilon$.

- We do not fix N in advance. Instead, we state our goal as follows:

Find a point $\bar{x} \in Q$ and a trust radius $r \geq \bar{r}$ such that

$$f(\bar{x}) - f(x^*(\eta r)) \leq \epsilon, \quad g(\bar{x}) \leq \epsilon, \quad \|\bar{x} - x_0\| \leq r,$$

where $\bar{r} > 0$ is the lower for the trust radius, and $\eta > 1$ is the safety factor. In order to find such a point, we skip in the method (4.9) the termination by the predefined number of step. Instead, we introduce the stopping criterion

$$kh^2 \geq \eta^2 R_k^2, \quad R_k^2 \stackrel{\text{def}}{=} \max\{\bar{r}^2, \|x_i - x_0\|^2, i = 0, \dots, k\} \quad (4.17)$$

This criterion can be justified by the third expression in the chain of inequalities (4.13).

2. Lipschitz constants $L(f)$, $L(g)$ are not known. Then we can start the method (4.9) with parameters based on the lower bounds for these constants. Each time we see that the norms of corresponding subgradients are bigger than these bounds, we multiply the bounds by $\eta > 1$ and restart the method from scratch. In accordance to (4.16), the length of the next stage will be in η^2 times longer. This means that the cost of the last stage is proportional to the total cost of all previous stages. On the other hand, our estimates will never exceed the actual Lipschitz constants more than in η times. For numerical experiments, we usually take $\eta = 2$.

The main advantage of methods (4.3) and (4.9) consists in their simplicity. If the basic feasible set Q is separable, then the sparse subgradients of the objective function and constraints results in a sparse update of the testing point. This allows us to apply a sparse updating technique of Section 2 for updating the matrix-vector products, which can be used for computing the new values and subgradients of the functional components. We consider the corresponding computational strategies in the next section.

5 Solving the huge-scale optimization problems

Huge-scale optimization problems are the problems of very big size, for which even the usual operations like addition of two vectors or matrix-vector multiplication become extremely time consuming. For solving such problems, we need to find an efficient way of treating their structure, especially the structure of linear operators.

In this section, we assume that the main operator $A(x) \stackrel{\text{def}}{=} Ax + b$ has a *block structure*. Namely, we assume that the matrix $A \in R^{M \times N}$ is divided on mn blocks $A_{i,j} \in R^{r_i \times q_j}$:

$$\sum_{i=1}^m r_i = M, \quad \sum_{j=1}^n q_j = N.$$

Accordingly, we partition the vectors of variables, and the vectors in the image space:

$$x = (x^1, \dots, x^n) \in R^N, \quad u = (u^1, \dots, u^m) \in R^M.$$

We assume that the block row $A_i = (A_{i,1}, \dots, A_{i,n})$ is block-sparse:

$$A_{i,j} \neq 0, \quad j \in \sigma_b(A_i) \subseteq \{1, \dots, n\}.$$

However, each block $A_{i,j}$ is considered as fully dense:

$$p(A_{i,j}) = q_i r_j, \quad j \in \sigma_b(A_i), \quad i = 1, \dots, m. \quad (5.1)$$

Similarly, we define the block column A^j of matrix A , $j = 1, \dots, n$:

$$A_{i,j} \neq 0, \quad i \in \sigma_b(A^j) \subseteq \{1, \dots, m\}.$$

Consider an optimization problem in the following form:

$$\begin{aligned} \min\{f(x) \stackrel{\text{def}}{=} f_0(u^0(x)) : \quad & \psi(u) \stackrel{\text{def}}{=} \max_{1 \leq i \leq m} f_i(u^i), \quad g(x) \stackrel{\text{def}}{=} \psi(u(x)) \leq 0, \\ & u^i(x) = \sum_{j \in \sigma_b(A_i)} A_{i,j} x^j - b^j, \quad i = 1, \dots, m, \\ & x^j \in Q_j, \quad j = 1, \dots, n\}, \end{aligned} \quad (5.2)$$

where convex functions $f_i(u^i)$, $i = 0, \dots, m$, have bounded subgradients, and the sets $Q_j \subseteq R^{q_j}$, $j = 1, \dots, n$, are closed and convex. Moreover, we assume that these sets are simple, presuming a possibility to compute corresponding Euclidean projections in $c_\pi q_j$ operations, where $c_\pi \geq 1$ is an absolute constant. Similarly, we assume that each function $f_i(\cdot)$ is simple, which means that its value and subgradient are computable in $c_f r_i$ operations, where $c_f \geq 1$ is an absolute constant.

In order to solve this problem by method (4.9), we need to implement efficiently the following operations:

- Computation of the value $g(x)$, vector $g'(x)$ and its Euclidean norm. Same computations for function $f(x)$.
- Implementation of the gradient step.
- Recursive update of the residual vector $u(x) = Ax - b$ for $x = x_{k+1}$.

In fact, it is better to implement all these operations in parallel by recursive updates.

Assume that we have already computed the direction d_k for updating the test point:

$$x_{k+1} = \pi_Q(x_k + d_k).$$

Assume also that d_k is block-sparse. Then this update can be done only for nonzero block components of d_k :

$$\begin{aligned} x_{k+1}^j &= \pi_{Q_j}(x_k^j + d_k^j), \quad j \in \sigma_b(d_k), \\ x_{k+1}^j &= x_k^j, \quad \text{otherwise.} \end{aligned} \quad (5.3)$$

Hence, the update of the image vector of the linear operator $A(x)$ is also sparse. Indeed, define $\delta_k^j = x_{k+1}^j - x_k^j$, $j \in \sigma_b(d_k)$. If the vector $u_k = Ax_k - b$ is already computed, then the vector $u_{k+1} = Ax_{k+1} - b$ can be obtained by a sequence of recursive updates. We

start by setting $u_+ = u_k$, and then implement the following operations:

For $j \in \sigma_b(d_k)$, $i \in \sigma_b(A^j)$ **iterate:**

1. Update $u_+^i = u_+^i + A_{i,j} \delta_k^j$.
2. Compute $f_i(u_+^i)$ and $f'_i(u_+^i)$.
3. Update the value $\psi(u_+)$ and $i_+ \stackrel{\text{def}}{=} \arg \max_{1 \leq l \leq m} f_l(u_+^l)$.

(5.4)

The final vector u_+ is accepted as u_{k+1} . Thus,

$$\begin{aligned} g(x_{k+1}) &= \psi(u_+), \quad g'(x_{k+1}) = A_{i_+}^T f'_{i_+}(u_{k+1}^{i_+}), \\ \|g'(x_{k+1})\|^2 &= \sum_{j \in \sigma_b(A_{i_+})} \|g'(x_{k+1})^j\|^2. \end{aligned} \quad (5.5)$$

Finally, if $u_+^0 \neq u_k^0$, and $g(x_{k+1}) \leq h \|g'(x_{k+1})\|$, then $i_+ = 0$ and we need to compute

$$f'(x) = A_0^T f'_0(u_+^0).$$

After these computations, everything is ready for making a final choice of the direction d_{k+1} , which will have the same block sparsity pattern as $A_{i_+}^T$.

Let us estimate the computational cost of this computational strategy, taking into account only the leading terms.

- The update of the test points (5.3) needs

$$\sum_{j \in \sigma_b(d_k)} c_\pi q_j = c_\pi p_b(d_k)$$

operations, provided that all of them are stored in the same array of computer memory.

- Each loop (5.4) needs $r_i q_j$ operations for Item 1, $c_f r_i$ operations for Item 2, and $\log_2 m$ operations for updating the maximum of m values by the technique presented in Section 3. Thus, the main term of the total complexity of the full loop is equal to

$$\sum_{j \in \sigma_b(d_k)} \sum_{i \in \sigma_b(A^j)} (r_i q_j + \log_2 m) = \sum_{j \in \sigma_b(d_k)} [p(A^j) + \log_2 m \cdot p_b(A^j)]. \quad (5.6)$$

operations.

- The computations (5.5) need $p(A_{i_+})$ operations.

Thus, the computational cost (5.6) dominates all other expenses. Consider two cases, when this cost is very small.

Theorem 3 Assume that the filling of matrix A is uniform:

$$\begin{aligned} \frac{1}{r_i} p(A_i) &\leq \frac{c_u}{M} p(A), \quad p_b(A_i) \leq \frac{c_u}{m} p_b(A), \quad i = 1, \dots, m, \\ \frac{1}{q_j} p(A^j) &\leq \frac{c_u}{N} p(A), \quad p_b(A^j) \leq \frac{c_u}{n} p_b(A), \quad j = 1, \dots, n, \end{aligned} \quad (5.7)$$

Then the computational cost (5.6) does not exceed

$$c_u^2 [\gamma^2(A) MN + \gamma_b^2(A) \cdot mn \cdot \log_2 m]. \quad (5.8)$$

Proof:

Indeed, in view of (5.7), the first term in the upper bound (5.6) can be estimated as follows:

$$\begin{aligned} \sum_{j \in \sigma_b(d_k)} p(A^j) &\leq \frac{c_u}{N} p(A) \sum_{j \in \sigma_b(d_k)} q_j = \frac{c_u}{N} p(A) \cdot \frac{1}{r_{i+}} p(A_{i+}) \\ &\leq \frac{c_u^2}{MN} p^2(A) = c_u^2 \gamma^2(A) MN. \end{aligned}$$

$$\text{Similarly, } \sum_{j \in \sigma_b(d_k)} p_b(A^j) \leq \frac{c_u}{n} p_b(A) \cdot p_b(A_{i+}) \leq \frac{c_u^2}{mn} p_b^2(A) = c_u^2 \gamma_b^2(A) mn. \quad \square$$

Note that both terms in the estimate (5.8) can be dominating. Indeed, consider the simple case

$$r_i = r, i = 1, \dots, m, \quad q_j = q, j = 1, \dots, n, \quad M = mr, \quad N = nq.$$

Then $p(A) = pq \cdot p_b(A)$. Therefore

$$\gamma(A) = \frac{p(A)}{MN} = \frac{p_b(A)}{mn} = \gamma_b(A).$$

Hence, the estimate (5.8) can be written as follows:

$$c_u^2 \gamma^2(A) mn [pq + \log_2 m].$$

In our second example, for the sake of notation, consider the case when $r_i \equiv 1$ and $q_j \equiv 1$. Let us assume that

$$p(A_i) \leq c_r, i = 1, \dots, n, \quad p(A^j) \leq c_q, j = 1, \dots, n, \quad (5.9)$$

where c_r and c_q are absolute constants. Then, the bound (5.6) can be estimated as follows:

$$(1 + \log_2 M) \sum_{j \in \sigma(d_k)} p(A^j) \leq (1 + \log_2 M) c_q p(A_{i+}) \leq (1 + \log_2 M) c_q c_r. \quad (5.10)$$

Thus, in such problems the computational cost of one iteration of subgradient method (4.9) grows *logarithmically* with dimension of the image space. Note that the condition (5.9) is satisfied in many practical applications, where the linear operators have only few nonzero diagonals (e.g. the problems related to finite elements or to partial differential equations). If this number is equal to q , then the conditions (5.9) are satisfied with $c_r = c_q = q$.

6 Random sparse block-coordinate methods

The simplest methods for solving huge-scale optimization problems are the coordinate descent schemes. The efficiency estimates for these methods as applied to the smooth optimization problems were first obtained in [5]. In this section we analyze a variant of these methods as applied to the class of nonsmooth optimization problems with sparse

block structure, which was described in Section 5. Namely, our problem of interest is as follows:

$$\min\{g(x) \stackrel{\text{def}}{=} \max_{1 \leq i \leq m} f_i(u^i), \quad u^i(x) = \sum_{j \in \sigma_b(A_i)} A_{i,j} x^j - b^i, \quad i = 1, \dots, m, \quad (6.1)$$

$$x^j \in Q_j, \quad j = 1, \dots, n\},$$

where convex functions $f_i(u^i)$, $i = 0, \dots, m$, have bounded subgradients, and the sets $Q_j \subseteq R^{q_j}$, $j = 1, \dots, n$, are closed and convex.

For each point $x \in R^N$, define an active index $i(x)$ such that $g(x) = f_{i(x)}(u^{i(x)}(x))$. Then

$$g'(x) = A_{i(x)}^T f'_{i(x)}(u^{i(x)}(x)), \quad \sigma_b(g'(x)) \subseteq \sigma_b(A_{i(x)}), \quad p_b(g'(x)) \leq p_b(A_{i(x)}).$$

For this point, define a random variable $\xi(x)$, which generates indexes from $\sigma_b(A_{i(x)})$ with equal probabilities $1/p_b(A_{i(x)})$. Assuming that the optimal value g^* of problem (6.1) is known, we can define now a random vector variable $\text{Next}(x)$ by the following rules:

1. Compute $h(x) = \frac{g(x) - g^*}{\|g'(x)\|^2}$. Generate $j(x) = \xi(x)$.
2. Define $[\text{Next}(x)]^{j(x)} = \pi_{Q_{j(x)}} \left(x - h(x) A_{i(x), j(x)}^T f'_{i(x)}(u^{i(x)}(x)) \right)$. (6.2)
3. For other indexes $j \neq j(x)$, define $[\text{Next}(x)]^j = x^j$.

Consider now a random block-coordinate variant of the method (4.3)

0. Choose $x_0 \in Q \equiv \prod_{j=1}^n Q_j$. Compute $u_0 = u(x_0)$ and $g(x_0)$.

1. k th iteration ($k \geq 0$).

a) Generate $j_k = \xi(x_k)$ and update $x_{k+1} = \text{Next}(x_k)$.

b) Update $u_{k+1} = u_k + A^{j_k} (x_{k+1}^{j_k} - x_k^{j_k})$, computing in parallel the values $f_i(u_{k+1}^i)$, $i \in \sigma_b(A^{j_k})$ with immediate evaluation of $g(x_{k+1})$.

(6.3)

Note that this method defines a sequence of discrete random variables $\{x_k\}$. Let us estimate its expected performance. Denote $g_k^* = \min_{0 \leq i \leq k} g(x_i)$.

Theorem 4 *Let $p_b(A_i) \leq r$, $i = 1, \dots, m$. Then, for any $k \geq 0$ we have:*

$$\mathcal{E}([g_k^* - g^*]^2) \leq \frac{rL^2(g)\|x_0 - \pi_{X_*}(x_0)\|^2}{k+1}, \quad (6.4)$$

$$\mathcal{E}(\|x_k - x_*\|^2) \leq \|x_0 - x_*\|^2, \quad \forall x_* \in X_*. \quad (6.5)$$

Proof:

Let us fix an arbitrary $x_* \in X_*$. Then for $x_{k+1}(x) \equiv \text{Next}(x)$.

$$\begin{aligned} \|x_{k+1}(x) - x_*\|^2 &\leq \|x^{j(x)} - x_*^{j(x)} - h(x)A_{i(x),j(x)}^T f'_{i(x)}(u^{i(x)}(x))\|^2 + \sum_{j \neq j(x)} \|x^j - x_*^j\|^2 \\ &= \|x - x_*\|^2 - 2h(x)\langle f'_{i(x)}(u^{i(x)}(x)), A_{i(x),j(x)}(x^{j(x)} - x_*^{j(x)}) \rangle \\ &\quad + h^2(x)\|A_{i(x),j(x)}^T f'_{i(x)}(u^{i(x)}(x))\|^2 \end{aligned}$$

Hence,

$$\begin{aligned} \mathcal{E}(\|x_{k+1}(x) - x_*\|^2 | x = x_k) &\leq \|x_k - x_*\|^2 \\ &\quad - \frac{2h(x_k)}{p_b(A_{i(x_k)})} \langle f'_{i(x_k)}(u^{i(x_k)}(x_k)), A_{i(x_k)}(x_k - x_*) \rangle + \frac{h^2(x_k)}{p_b(A_{i(x_k)})} \|A_{i(x_k)}^T f'_{i(x_k)}(u^{i(x_k)}(x_k))\|^2 \\ &\leq \|x_k - x_*\|^2 - \frac{2h(x_k)}{p_b(A_{i(x_k)})} (g(x_k) - g^*) + \frac{h^2(x_k)}{p_b(A_{i(x_k)})} \|g(x_k)\|^2 \\ &= \|x_k - x_*\|^2 - \frac{(g(x_k) - g^*)^2}{p_b(A_{i(x_k)}) \|g(x_k)\|^2}. \end{aligned}$$

Taking now the full expectation, we get

$$\mathcal{E}(\|x_{k+1} - x_*\|^2) \leq \mathcal{E}(\|x_k - x_*\|^2) - \frac{1}{rL^2(g)} \mathcal{E}((g(x_k) - g^*)^2).$$

This proves inequality (6.5). Summing up these inequalities, we have

$$\begin{aligned} \|x_0 - x_*\|^2 &\geq \frac{1}{rL^2(g)} \sum_{i=0}^k \mathcal{E}((g(x_i) - g^*)^2) \\ &\geq \frac{1}{rL^2(g)(k+1)} \mathcal{E}\left(\left[\sum_{i=0}^k (g(x_i) - g^*)\right]^2\right) \geq \frac{k+1}{rL^2(g)} \mathcal{E}([g_k^* - g^*]^2), \end{aligned}$$

which gives us (6.4). $\square$

Let us estimate the computational cost of one iteration of method (6.3). For the sake of simplicity, assume that $r_i \equiv q_j \equiv 1$, and all $p(A^j) \leq q$. In this case, its computational cost (which is mainly related to Step b)) is as follows:

$$\log_2 m \cdot p_b(A^{j_k}) + \sum_{i \in \sigma_b(A^{j_k})} r_i q_{j_k} = (1 + \log_2 m) p(A^{j_k}) \leq (1 + \log_2 m) q.$$

operations. Hence, under assumption of Theorem 4, we get $\mathcal{E}([g_k^* - g^*]^2) \leq \epsilon^2$ in $O\left(\frac{qrL^2(g)\|x_0 - x_*\|^2}{\epsilon^2} \log_2 m\right)$ operations.

Let us describe a block-coordinate version of method (4.9) as applied to problem (5.2). Let us fix a step size $h > 0$. For any $x \in Q = \prod_{j=1}^n Q_j$ define an indicator $I(x)$ as follows:

$$I(x) = \begin{cases} i(x), & \text{if } g(x) > h\|g'(x)\|, \\ 0, & \text{otherwise.} \end{cases} \quad (6.6)$$

Now we can define a random variable $\bar{\xi}(x)$, which generates indexes from $\sigma_b(A_{I(x)})$ with equal probabilities $1/p_b(A_{I(x)})$. The random vector variable $\text{Next}_C(x)$ is defined now as follows:

1. If $I(x) > 0$, then $h(x) = \frac{g(x)}{\|g'(x)\|^2}$, else $h(x) = h$. Generate $j(x) = \bar{\xi}(x)$.
2. Define $[\text{Next}_C(x)]^{j(x)} = \pi_{Q_{j(x)}} \left(x - h(x) A_{I(x), j(x)}^T f'_{I(x)}(u^{I(x)}(x)) \right)$. (6.7)
3. For other indexes $j \neq j(x)$, define $[\text{Next}_C(x)]^j = x^j$.

Using this operation in the scheme (6.3), we get a random block-coordinate method for problem (5.2). The complexity analysis of this scheme can be easily done by combining the arguments from Theorem 2 and 4.

7 Computational experiments: Google problem

In this section, we check the practical performance of method (4.3). Our experiments were performed on a standard PC with 2.6GHz processor and 4Gb of RAM.

From the above theoretical analysis, we know that the sparse updating technique can be efficient if $\gamma(A) \ll 1 \ll MN$. This condition is satisfied by the *Google Problem*.

Let $E \in R^{N \times N}$ be an incidence matrix of a graph. Denote $e = (1, \dots, 1)^T \in R^N$ and

$$\bar{E} = E \cdot \text{diag}(E^T e)^{-1}.$$

Since, $\bar{E}^T e = e$, this matrix is stochastic. Our problem consists in finding a maximal right eigenvector of the matrix $\bar{E}$:

$$\text{Find } x^* \geq 0 : \quad \bar{E}x^* = x^*, \quad x^* \neq 0.$$

In [5] it was suggested to solve this problem by applying a random coordinate descent method to the quadratic function

$$\frac{1}{2} \|\bar{E}x - x\|^2 + \frac{\gamma}{2} [\langle e, x \rangle - 1]^2, \quad (7.1)$$

where $\gamma > 0$ is a penalty parameter for a normalizing linear equality constraint. If the degree of each node in the graph is small, then the computation of partial derivatives of this function is cheap. Hence, the technique [5] can be applied to the matrices of very big size.

Note that the maximal eigenvalue of a nonnegative matrix $A \in R^{n \times n}$ satisfies the condition

$$\rho(A) = \min_{x \geq 0} \max_{1 \leq i \leq n} \frac{1}{x^{(i)}} \langle A^T e_i, x \rangle.$$

Moreover, the minimum is attained at the corresponding eigenvector. Since in our situation $\rho(\bar{E}) = 1$, we can characterize x^* as a solution to the following convex nonsmooth minimization problem:

$$g(x) \stackrel{\text{def}}{=} \max_{1 \leq i \leq N} [\langle A^T e_i, x \rangle - x^{(i)}] \rightarrow \min_{x \geq 0}. \quad (7.2)$$

Note that $g^* = 0$, and the optimal set X^* of this problem is a convex cone. This feature suggests to solve (7.2) by method (4.3). Let us choose for this method the starting point $x_0 = e$. Then the whole minimizing sequence $\{x_k\}$ constructed by (4.3), is well separated from zero. Indeed, for any $x^* \in X^*$ the second inequality in (4.6) implies that

$$\langle x^*, e \rangle \leq \langle x^*, x_k \rangle \leq \|x^*\|_1 \cdot \|x_k\|_\infty = \langle x^*, e \rangle \cdot \|x_k\|_\infty.$$

Thus, our goal will be as follows:

$$\text{Find } \bar{x} \geq 0 \text{ such that } \|\bar{x}\|_\infty \geq 1 \text{ and } g(\bar{x}) \leq \epsilon. \quad (7.3)$$

Recall that the first condition is satisfied by method (4.3) automatically.

In our test problem we generate a random graph with uniform degree $p \geq 2$ for each node. Thus, the number of elements in each column of matrix A is exactly p . The number of elements in the rows can be different, but still it is p in average. Hence, from the analysis of example (5.9), we can expect that the complexity of one iteration of the subgradient method (4.3) is of the order

$$O(p^2 \ln N) \quad (7.4)$$

operations. An upper estimate for the number of iterations of (4.3) can be derived from (4.4). Since X_* is a cone, we have

$$\text{dist}^2(X_*, e) \leq N - 1.$$

Since

$$\|A^T e_j\|^2 = p \times \frac{1}{p^2} = \frac{1}{p},$$

The upper bound for the number of iterations is

$$\frac{L^2(f)}{\epsilon^2} \|x_0 - \pi_{X_*}(x_0)\|^2 \leq \frac{2N}{p\epsilon^2}. \quad (7.5)$$

Thus, the total computational cost of solving this problem by method (4.3) does not exceed

$$O\left(pN \cdot \frac{\ln N}{\epsilon^2}\right) \quad (7.6)$$

operations. For the subgradient method, the estimate for the number of iterations will be the same, but the cost of one iteration is of the order $O(pN)$ operations. Thus, it needs $O\left(\frac{N^2}{\epsilon^2}\right)$ operations. Finally, the smoothing technique [3] needs $O\left(\frac{L(f)}{\epsilon} \|x_0 - \pi_{X_*}(x_0)\|\right)$ iterations of the gradient-type scheme. Thus, its total cost is of the order

$$O\left(\frac{N^{1/2}}{p^{1/2}\epsilon} \cdot pN\right)$$

operations. We can see that only the sparse subgradient method (4.3) has practically linear dependence of the computational cost in the dimension N .

Let us present now the results of our numerical experiments. First of all, let us compare the growth of computational time of the method (4.3) with the *sparse updates* (SUSM) and the same method (SM) with the *sparse computation* of the matrix-vector product (with complexity (2.1)). In the table below, we put the measurements for both methods as applied to the random Google problem (7.2). In the first column of the table we put

the dimension of the matrix, the second column displays its row capacity (2.8), and the last two columns show the computational time (in seconds) for the new method and for the standard one.

Time for 10^4 iterations ($p = 32$)

N	$\kappa(A)$	SUSM	SM
1024	1632	3.00	2.98
2048	1792	3.36	6.41
4096	1888	3.75	15.11
8192	1920	4.20	139.92
16384	1824	4.69	408.38

(7.7)

Since the row capacity in these examples is almost constant, the computational time of SUSM almost does not grow. The computational time of SM grows linearly, and after even quadratically with dimension. The later effect can be explained by the troubles with keeping massive computations in the fast memory of Windows system. For SUSM the required size of the fast memory is $O(\kappa(A))$, and for SM it is $O(N)$. Recall that the both schemes implement the same minimization method (4.3). The only difference between them consists in the way of computing/updating the matrix-vector products.

For the larger dimensions, the linear growth of the iteration cost in SM is restored. However, as compared with SUSM, it remains extremely slow.

Time for 10^3 iterations ($p = 16$)

N	$\kappa(A)$	SUSM	SM
131072	576	0.19	213.9
262144	592	0.25	477.8
524288	592	0.32	1095.5
1048576	608	0.40	2590.8

(7.8)

Therefore, in the sequel we look at the performance of SUSM only. In the next table we present the computational results for the problem (7.1) with $N = 131072$, $p = 16$, $\kappa(A) = 576$, and $L(f) = 0.21$.

Iterations	$g - g^*$	Time (sec)
$1.0 \cdot 10^5$	0.1100	16.44
$3.0 \cdot 10^5$	0.0429	49.32
$6.0 \cdot 10^5$	0.0221	98.65
$1.1 \cdot 10^6$	0.0119	180.85
$2.2 \cdot 10^6$	0.0057	361.71
$4.1 \cdot 10^6$	0.0028	674.09
$7.6 \cdot 10^6$	0.0014	1249.54
$1.0 \cdot 10^7$	0.0010	1644.13

(7.9)

Despite to a large dimension and sufficiently high accuracy, the computational time is still reasonable. Finally, let us present the results for a big random problem with $N = 1048576$,

$p = 8$, $\kappa(A) = 192$, and $L(f) = 0.21$.

Iterations	$g - g^*$	Time (sec)
0	2.000000	0.00
$1.0 \cdot 10^5$	0.546662	7.69
$4.0 \cdot 10^5$	0.276866	30.74
$1.0 \cdot 10^6$	0.137822	76.86
$2.5 \cdot 10^6$	0.063099	192.14
$5.1 \cdot 10^6$	0.032092	391.97
$9.9 \cdot 10^6$	0.016162	760.88
$1.5 \cdot 10^7$	0.010009	1183.59

(7.10)

The size of the final point $\bar{x}_*$ of this process is as follows:

$$\|\bar{x}_*\|_\infty = 2.941497, \quad R_0^2 \stackrel{\text{def}}{=} \|\bar{x}_* - e\|_2^2 = 1.2 \cdot 10^5.$$

Thus, the upper bound for the number of iterations for achieving the accuracy $\epsilon = 10^{-2}$ by method (4.3) is $\frac{L^2(f)R_0^2}{\epsilon^2} = 5.3 \cdot 10^7$. Note that this estimate is very close to the results shown in (7.10). However, for the problems of this size, the total computational time remains quite reasonable due to the efficient sparse matrix update. From the table (7.8) we can guess that for the same number of iterations, the usual subgradient method would require almost a year of computations.

Finally, let us present computational results for method (6.3) as applied to the problem (7.2) with the same parameters as in (7.10).

Iterations	$g - g^*$	Time (sec)
0	2.00000	0
$1.0 \cdot 10^6$	0.55124	7.04
$5.0 \cdot 10^6$	0.27433	35.22
$1.4 \cdot 10^7$	0.12881	98.62
$3.4 \cdot 10^7$	0.05628	239.50
$5.9 \cdot 10^7$	0.03162	415.60
$1.1 \cdot 10^8$	0.01636	795.97
$1.6 \cdot 10^8$	0.01006	1148.17

(7.11)

Despite to the very different number of iterations, we can see that the performance of both methods is quite similar.

8 Discussion

As we have seen, if the main linear operator A of the optimization problem has small row density, then the sparse updating strategy transforms a standard slow subgradient method in a very efficient scheme. Unfortunately, even a single dense row of matrix A usually results in a big value of $\kappa(A)$. Let us show that this situation can be treated by introducing into the problem some additional variables.

Assume, for example, that row i of matrix $A \in R^{m \times n}$ is dense. Note that linear inequality

$$\langle A^T e_i, x \rangle \leq b^{(i)}, \quad x \in R^n, \quad (8.1)$$

is equivalent to the following *system* of linear equalities and inequalities:

$$\begin{aligned} y^{(1)} &= A_{i,1} x^{(1)}, \\ y^{(j)} &= y^{(j-1)} + A_{i,j} x^{(j)}, \quad j = 2, \dots, n, \\ y^{(n)} &\leq b^{(i)}. \end{aligned} \tag{8.2}$$

We call this system an *extended representation* of i th row of matrix A .

Note that we need to introduce *new variables* $y^{(j)}$ only for nonzero coefficients of vector a . Thus, for rewriting the whole system of linear inequalities $Ax \leq b$ in the extended form, we need to introduce $p(A)$ additional variables and $p(A)$ additional equality constraints. However, the complexity bound (5.10) of each iteration of sparse update depends *logarithmically* on the number of constraints. At the same time, the extended system has at most three variables in each row. Thus, if the columns of matrix A are not too dense, we can get very efficient sparse updating procedure.

In the above approach there is a hidden drawback. If we solve an optimization problem with extended representation of the linear operator, then inequalities (8.2) are treated as the *constraints* of the problem. Hence, in the end we can guarantee that they are satisfied with accuracy $\epsilon > 0$. This means, that the value $y^{(n)}$ differs from $\langle A^T e_i, x \rangle$ by $n\epsilon$ (we assume that $p(A^T e_i) = n$). It seems that this *accumulation of errors* destroys all advantages of the proposed treatment.

However, the situation is not so bad. Indeed, in many problems it is natural to relate the feasible error in a linear inequality with the number of its nonzero coefficients. Then the above technique automatically delivers the solutions with correct level of accuracy.

A similar technique can be applied to the *dense columns* of matrix A . If column Ae_j is dense, then for each nonzero coefficient $A_{i,j}$ we introduce a new variable $z_{i,j}$ and replace the term $A_{i,j}x^{(j)}$ in the product Ax by $A_{i,j}z_{i,j}$. It remains to add $p(Ae_j)$ linear equality constraints:

$$z_{i,1} = \dots = x^{(j)},$$

(which contain only the indexes of nonzero coefficients).

References

- [1] L.Khachiyan, S.Tarasov, and E.Erich. The inscribed ellipsoid method. Soviet Math. Dokl., **298** (1988).
- [2] Z.Q. Luo, P. Tseng. On the convergence rate of dual ascent methods for linearly constrained convex minimization. *Math. of Operations Research*, **18**(2), 846-867 (1993).
- [3] Yu. Nesterov. Smooth minimization of non-smooth functions, *Mathematical Programming* (A), **103** (1), 127-152 (2005).
- [4] Yu. Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical Programming*, **120**(1), 261-283 (August 2009)
- [5] Yu. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. CORE Discussion Paper 2010/2. Accepted by *SIOPT*.

- [6] Yu. Nesterov, A. Nemirovskii. *Interior point polynomial methods in convex programming: Theory and Applications*, SIAM, Philadelphia, 1994.
- [7] A. Gilpin, J. Pena, and T. Sandholm. First-order algorithm with $O(\ln(1/\epsilon))$ convergence for ϵ -equilibrium in two-person zero-sum games. To appear in *Mathematical Programming*.
- [8] B. Polyak. Introduction to Optimization. Optimization Software, Inc., 1987.
- [9] P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. April 2011 (revised July 4, 2011); submitted to *Mathematical Programming*.
- [10] N. Shor. Minimization Methods for Non-differentiable Functions. Springer Series in Computational Mathematics. Springer, 1985.