Visualizing Multidimensional Linear Programming Problems

Nikolay A. Olkhovsky and Leonid B. Sokolinsky(✉)

South Ural State University (National Research University),
76, Lenin prospekt, Chelyabinsk 454080, Russia
{olkhovskiiNA,leonid.sokolinsky}@susu.ru

Abstract. The article proposes an n-dimensional mathematical model of the visual representation of a linear programming problem. This model makes it possible to use artificial neural networks to solve multidimensional linear optimization problems, the feasible region of which is a bounded non-empty set. To visualize a linear programming problem, an objective hyperplane is introduced, its orientation is determined by the gradient of the linear objective function: the gradient is the normal to the objective hyperplane. In the case of searching the maximum, the objective hyperplane is positioned in such a way that the value of the objective function at all its points exceeds the value of the objective function at all points of the feasible region, which is a bounded convex polytope. For an arbitrary point of the objective hyperplane, the objective projection onto the polytope is determined: the closer the objective projection point is to the objective hyperplane, the greater the value of the objective function at this point. Based on the objective hyperplane, a finite regular set of points, called the receptive field, is constructed. Using objective projections, an image of the polytope is constructed. This image includes the distances from the receptive points to the corresponding points of the polytope surface. Based on the proposed model, parallel algorithms for visualizing a linear programming problem are constructed. An analytical estimation of its scalability is performed. Information about the software implementation and the results of large-scale computational experiments confirming the efficiency of the proposed approaches are presented.

Keywords: Linear programming · Multydimensional visualization · Mathematical model · Parallel algorithm · BSF-skeleton

1 Introduction

The rapid development of Big Data technologies [11,12] has led to the emergence of mathematical optimization models in the form of large-scale linear programming (LP) problems [24]. Such problems arise in industry, economics, logistics, statistics, quantum physics, and other fields [3,4,8,22,25]. In many cases, the conventional software is not able to handle such large-scale LP problems in an

L. Sokolinsky and M. Zymbler (Eds.): PCT 2022, CCIS 1618, pp. 172–196, 2022.
https://doi.org/10.1007/978-3-031-11623-0_13

acceptable time [2]. At the same time, in the nearest future, exascale supercomputers potentially capable of solving such problems will appear [6]. In accordance with this, the issue of developing new effective methods for solving large-scale LP problems using exascale supercomputing systems is urgent.

Until now, the class of algorithms proposed and developed by Dantzig on the basis of the simplex method [5] is one of the most common ways to solve LP problems. The simplex method is effective for solving a large class of LP problems. However, the simplex method has some fundamental features that limit its applicability to large LP problems. First, in the worst case, the simplex method traverses all the vertices of the simplex, which results in exponential time complexity [35]. Second, in most cases, the simplex method successfully solves LP problems containing up to 50,000 variables. However, a loss of precision is observed when the simplex method is used for solving large LP problems. Such a loss of precision cannot be compensated even by applying such computational intensive procedures as "affine scaling" or "iterative refinement" [34]. Third, the simplex method does not scale well on multiprocessor systems with distributed memory. Many attempts to parallelize the simplex method were made, but they all failed [19]. In [14], Karmarkar proposed the inner point method having polynomial time complexity in all cases. This method effectively solves problems with millions of variables and millions of constraints. Unlike the simplex method, the inner point method is self-correcting. Therefore, it is robust to the loss of precision in computations. The drawbacks of the interior point method are as follows. First, the interior point method requires the careful tuning of its parameters. Second, this method needs a known point that belongs to the feasible region of the LP problem to start calculations. Finding such an interior point can be reduced to solving an additional LP problem. An alternative is iterative projection-type methods [23,26,31], which are also self-correcting. Third, like the simplex method, the inner point method does not scale well on multiprocessor systems with distributed memory. Several attempts at effective parallelization for particular cases were made (see, for example, [10,15]). However, it was not possible to make efficient parallelization for the general case. In accordance with this, research directions related to the development of new scalable methods for solving LP problems are urgent.

A possible efficient alternative to the conventional methods of LP is optimization methods based on neural network models. Artificial neural networks [20,21] are one of the most promising and rapidly developing areas of modern information technology. Neural networks are a universal tool capable of solving problems in almost all areas. The most impressive success was achieved in image recognition and analysis using convolutional neural networks [18]. However, in scientific periodicals, there are almost no works devoted to the use of convolutional neural networks for solving linear optimization problems [17]. The reason is that convolutional neural networks focus on image processing, but there are no works on the visual representation of multidimensional linear programming problems in the scientific literature. Thus, the issue of developing new neural network models and methods focused on linear optimization remains open.

In this paper, we try to develop an n-dimensional mathematical model of the visual representation of the LP problem. This model allows one to employ

the technique of artificial neural networks to solve multidimensional linear optimization problems, the feasible region of which is a bounded non-empty set. The visualization method based on the described model has high computational complexity. For this reason, we propose its implementation as a parallel algorithm designed for cluster computing systems. The rest of the paper is organized as follows. Section 2 is devoted to the design of the mathematical model of the visual representation of multidimensional LP problems. Section 3 describes the implementation of the proposed visualization method as a parallel algorithm and provides an analytical estimation of its scalability. Section 4 presents information about the software implementation of the described parallel algorithm and discusses the results of large-scale computational experiments on a cluster computing system. Section 5 summarizes the obtained results and provides directions for further research.

2 Mathematical Model of the LP Visual Representation

The linear optimization problem can be stated as follows

$$\bar{x} = \arg\max \left\{ \langle c, x \rangle \middle| \, Ax \leqslant b, x \in \mathbb{R}^n \right\}, \tag{1}$$

where $c, b \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $c \neq \mathbf{0}$. Here and below, $\langle \cdot \, , \cdot \rangle$ stands for the dot product of vectors. We assume that the constraint $x \geqslant \mathbf{0}$ is also included in the system $Ax \leqslant b$ in the form of the following inequalities:

$$\begin{array}{ccccccccccc}
-x_1 & + & 0 & + & \cdots\cdots\cdots & + & 0 & \leqslant & 0; \\
0 & - & x_2 & + & 0 & + \cdots + & 0 & \leqslant & 0; \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & + & \cdots\cdots\cdots & + & 0 & - & x_n & \leqslant & 0.
\end{array}$$

The vector c is the gradient of the linear objective function

$$f(x) = c_1 x_1 + \ldots + c_n x_n. \tag{2}$$

Let M denote the feasible region of problem (1):

$$M = \left\{ x \in \mathbb{R}^n \middle| \, Ax \leqslant b \right\}. \tag{3}$$

We assume from now on that M is a non-empty bounded set. This means that M is a convex closed polytope in the space $\mathbb{R}^n$, and the solution set of problem (1) is not empty.

Let $\tilde{a}_i \in \mathbb{R}^n$ be a vector formed by the elements of the ith row of the matrix A. Then, the matrix inequality $Ax \leqslant b$ is represented as a system of inequalities

$$\langle \tilde{a}_i, x \rangle \leqslant b_i, i = 1, \ldots, m. \tag{4}$$

We assume from now on that

$$\tilde{a}_i \neq \mathbf{0}. \tag{5}$$

for all $i = 1, \ldots, m$. Let us denote by H_i the hyperplane defined by the equation

$$\langle \tilde{a}_i, x \rangle = b_i \ (1 \leqslant i \leqslant m). \tag{6}$$

Thus,

$$H_i = \{x \in \mathbb{R}^n | \, \langle \tilde{a}_i, x \rangle = b_i\}. \tag{7}$$

Definition 1. *The half-space H_i^+ generated by the hyperplane H_i is the half-space defined by the equation*

$$H_i^+ = \{x \in \mathbb{R}^n | \, \langle \tilde{a}_i, x \rangle \leqslant b_i\}. \tag{8}$$

From now on, we assume that problem (1) is non-degenerate, i.e.,

$$\forall i \neq j \ : H_i \neq H_j \ (i, j \in \{1, \ldots, m\}). \tag{9}$$

Definition 2. *The half-space H_i^+ generated by the hyperplane H_i is recessive with respect to the vector c if*

$$\forall x \in H_i, \forall \lambda \in \mathbb{R}_{>0} \ : x - \lambda c \in H_i^+ \wedge x - \lambda c \notin H_i. \tag{10}$$

In other words, the ray coming from the hyperplane H_i in the direction opposite to the vector c lies completely in H_i^+, but not in H_i.

Proposition 1. *The necessary and sufficient condition for the recessivity of the half-space H_i^+ with respect to the vector c is the condition*

$$\langle \tilde{a}_i, c \rangle > 0. \tag{11}$$

Proof. Let us prove the necessity first. Let condition (10) hold. Equation (7) implies

$$x = \frac{b_i \tilde{a}_i}{\|\tilde{a}_i\|^2} \in H_i. \tag{12}$$

By virtue of (5),

$$\lambda = \frac{1}{\|\tilde{a}_i\|^2} \in \mathbb{R}_{>0}. \tag{13}$$

Comparing (10) with (12) and (13), we obtain

$$\frac{b_i \tilde{a}_i}{\|\tilde{a}_i\|^2} - \frac{1}{\|\tilde{a}_i\|^2} c \in H_i^+;$$
$$\frac{b_i \tilde{a}_i}{\|\tilde{a}_i\|^2} - \frac{1}{\|\tilde{a}_i\|^2} c \notin H_i.$$

In view of (7) and (8), this implies

$$\left\langle \tilde{a}_i, \tfrac{b_i \tilde{a}_i}{\|\tilde{a}_i\|^2} - \tfrac{1}{\|\tilde{a}_i\|^2} c \right\rangle < b_i. \tag{14}$$

Using simple algebraic transformations of inequality (14), we obtain (11). Thus, the necessity is proved.

Let us prove the sufficiency by contradiction. Assume that (11) holds, and there are $x \in H_i$ and $\lambda > 0$ such that

$$x - \lambda c \notin H_i^+ \vee x - \lambda c \in H_i.$$

In accordance with (7) and (8), this implies

$$\langle \tilde{a}_i, x - \lambda c \rangle \geqslant b_i$$

that is equivalent to

$$\langle \tilde{a}_i, x \rangle - \lambda \langle \tilde{a}_i, c \rangle \geqslant b_i.$$

Since $\lambda > 0$, it follows from (11) that

$$\langle \tilde{a}_i, x \rangle > b_i,$$

but this contradicts our assumption that $x \in H_i$. □

Definition 3. *Fix a point $z \in \mathbb{R}^n$ such that the half-space*

$$H_c^+ = \{x \in \mathbb{R}^n | \langle c, x - z \rangle \leqslant 0\} \tag{15}$$

includes the polytope M:

$$M \subset H_c^+.$$

In this case, we call the half-space H_c^+ the objective half-space, and the hyperplane H_c, defined by the equation

$$H_c = \{x \in \mathbb{R}^n | \langle c, x - z \rangle = 0\}, \tag{16}$$

the objective hyperplane.

Denote by $\pi_c(x)$ the *orthogonal projection* of the point x onto the objective hyperplane H_c:

$$\pi_c(x) = x - \frac{\langle c, x - z \rangle}{\|c\|^2} c. \tag{17}$$

Here, $\|\cdot\|$ stands for the Euclidean norm. Define the *distance* $\rho_c(x)$ from $x \in H_c^+$ to the objective hyperplane H_c as follows:

$$\rho_c(x) = \|\pi_c(x) - x\|. \tag{18}$$

Comparing (15), (17) and (18), we find that, in this case, the distance $\rho_c(x)$ can be calculated as follows:

$$\rho_c(x) = \frac{\langle c, z - x \rangle}{\|c\|}. \tag{19}$$

The following Proposition 2 holds.

Proposition 2. *For all* $x, y \in H_c^+$,

$$\rho_c(x) \leqslant \rho_c(y) \Leftrightarrow \langle c, x\rangle \geqslant \langle c, y\rangle .$$

Proof. Equation (19) implies that

$$\begin{aligned}
&\rho_c(x) \leqslant \rho_c(y) \Leftrightarrow \frac{\langle c, z-x\rangle}{\|c\|} \leqslant \frac{\langle c, z-y\rangle}{\|c\|} \\
&\Leftrightarrow \langle c, z-x\rangle \leqslant \langle c, z-y\rangle \\
&\Leftrightarrow \langle c, z\rangle + \langle c, -x\rangle \leqslant \langle c, z\rangle + \langle c, -y\rangle \\
&\Leftrightarrow \langle c, -x\rangle \leqslant \langle c, -y\rangle \\
&\Leftrightarrow \langle c, x\rangle \geqslant \langle c, y\rangle .
\end{aligned}$$

□

Proposition 2 says that problem (1) is equivalent to the following problem:

$$\bar{x} = \arg\min \{\rho_c(x) | \, x \in M\} . \tag{20}$$

Definition 4. *Let the half-space* H_i^+ *be recessive with respect to the vector* c. *The objective projection* $\gamma_i(x)$ *of the point* $x \in \mathbb{R}^n$ *onto the recessive half-space* H_i^+ *is a point defined by the equation*

$$\gamma_i(x) = x - \sigma_i(x)c, \tag{21}$$

where

$$\sigma_i(x) = \min \{\sigma \in \mathbb{R}_{\geqslant 0} \mid \; x - \sigma c \in H_i^+\} .$$

Examples of objective projections in $\mathbb{R}^2$ are shown in Fig. 1.

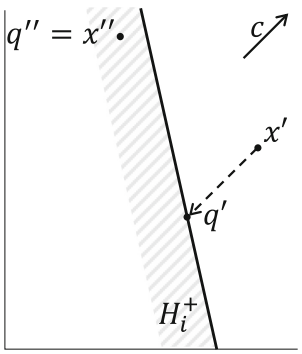

Fig. 1. Objective projections in the space $\mathbb{R}^2$: $\gamma_i(x') = q'$; $\gamma_i(x'') = q'' = x''$.

The following Proposition 3 provides an equation for calculating the objective projection onto a half-space that is recessive with respect to the vector c.

Proposition 3. *Let the half-space H_i^+ defined by the inequality*

$$\langle \tilde{a}_i, x \rangle \leqslant b_i \tag{22}$$

be recessive with respect to the vector c. Let

$$g \notin H_i^+. \tag{23}$$

Then,

$$\gamma_i(g) = g - \frac{\langle \tilde{a}_i, g \rangle - b_i}{\langle \tilde{a}_i, c \rangle} c. \tag{24}$$

Proof. According to Definition 4, we have

$$\gamma_i(g) = g - \sigma_i(g)c,$$

where

$$\sigma_i(x) = \min \left\{ \sigma \in \mathbb{R}_{\geqslant 0} \mid x - \sigma c \in H_i^+ \right\}.$$

Thus, we need to prove that

$$\frac{\langle \tilde{a}_i, g \rangle - b_i}{\langle \tilde{a}_i, c \rangle} = \min \left\{ \sigma \in \mathbb{R}_{\geqslant 0} \mid x - \sigma c \in H_i^+ \right\}. \tag{25}$$

Consider the strait line L defined by the parametric equation

$$L = \left\{ g + \tau c \middle| \, \tau \in \mathbb{R} \right\}.$$

Let the point q be the intersection of the line L with the hyperplane H_i:

$$q = L \cap H_i. \tag{26}$$

Then, q must satisfy the equation

$$q = g + \tau' c \tag{27}$$

for some $\tau' \in \mathbb{R}$. Substitute the right side of Eq. (27) into Eq. (6) instead of x:

$$\langle \tilde{a}_i, g + \tau' c \rangle = b_i.$$

It follows that

$$\begin{aligned} &\langle \tilde{a}_i, g \rangle + \tau' \langle \tilde{a}_i, c \rangle = b_i, \\ &\tau' = \frac{b_i - \langle \tilde{a}_i, g \rangle}{\langle \tilde{a}_i, c \rangle}. \end{aligned} \tag{28}$$

Substituting the right side of Eq. (28) into Eq. (27) instead of τ', we obtain

$$q = g + \frac{b_i - \langle \tilde{a}_i, g \rangle}{\langle \tilde{a}_i, c \rangle} c,$$

which is equivalent to

$$q = g - \frac{\langle \tilde{a}_i, g \rangle - b_i}{\langle \tilde{a}_i, c \rangle} c. \tag{29}$$

Since, according to (26), $q \in H_i$, Eq. (25) will hold if

$$\forall \sigma \in \mathbb{R}_{>0} : \sigma < \frac{\langle \tilde{a}_i, g \rangle - b_i}{\langle \tilde{a}_i, c \rangle} \Rightarrow g - \sigma c \notin H_i^+ \tag{30}$$

holds. Assume the opposite, i.e., there exist $\sigma' > 0$ such that

$$\sigma' < \frac{\langle \tilde{a}_i, g \rangle - b_i}{\langle \tilde{a}_i, c \rangle} \tag{31}$$

and

$$g - \sigma' c \in H_i^+. \tag{32}$$

Then, it follows from (22) and (32) that

$$\langle \tilde{a}_i, g - \sigma' c \rangle \leqslant b_i.$$

This is equivalent to

$$\langle \tilde{a}_i, g \rangle - b_i \leqslant \sigma' \langle \tilde{a}_i, c \rangle . \tag{33}$$

Proposition 1 implies that $\langle \tilde{a}_i, c \rangle > 0$. Hence, Eq. (33) is equivalent to

$$\sigma' \geqslant \frac{\langle \tilde{a}_i, g \rangle - b_i}{\langle \tilde{a}_i, c \rangle}.$$

Thus, we have a contradiction with (31). □

Definition 5. *Let $g \in H_c$. The* objective projection $\gamma_M(g)$ *of the point g onto the polytope M is a point defined by the following equation:*

$$\gamma_M(g) = g - \sigma_M(g) c, \tag{34}$$

where

$$\sigma_M(g) = \min \{ \sigma \in \mathbb{R}_{\geqslant 0} | \, g - \sigma c \in M \} .$$

If

$$\neg \exists \, \sigma \in \mathbb{R}_{\geqslant 0} : g - \sigma c \in M,$$

then we set $\gamma_M(g) = \vec{\infty}$, where $\vec{\infty}$ stands for a point that is infinitely far from the polytope M.

Examples of objective projections onto the polytope M in $\mathbb{R}^2$ are shown in Fig. 2.

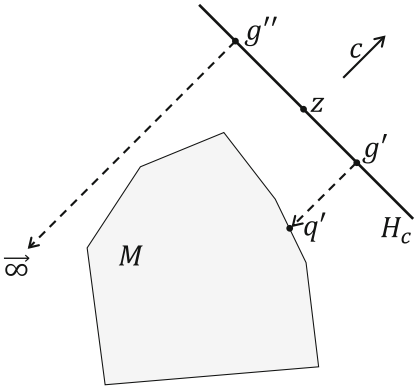

Fig. 2. Objective projections onto the polytope M in $\mathbb{R}^2$: $\gamma_M(g') = q'$; $\gamma_M(g'') = \vec{\infty}$.

Definition 6. *The receptive field* $\mathfrak{G}(z, \eta, \delta) \subset H_c$ *of the density* $\delta \in \mathbb{R}_{>0}$ *with the center* $z \in H_c$ *and the rank* $\eta \in \mathbb{N}$ *is a finite ordered set of points satisfying the following conditions:*

$$z \in \mathfrak{G}(z, \eta, \delta); \tag{35}$$

$$\forall g \in \mathfrak{G}(z, \eta, \delta) : \|g - z\| \leqslant \eta\delta\sqrt{n}; \tag{36}$$

$$\forall g', g'' \in \mathfrak{G}(z, \eta, \delta) : g' \neq g'' \Rightarrow \|g' - g''\| \geqslant \delta; \tag{37}$$

$$\forall g' \in \mathfrak{G}(z, \eta, \delta)\ \exists g'' \in \mathfrak{G}(z, \eta, \delta) : \|g' - g''\| = \delta; \tag{38}$$

$$\forall x \in \mathrm{Co}(\mathfrak{G}(z, \eta, \delta))\ \exists g \in \mathfrak{G}(z, \eta, \delta) : \|g - x\| \leqslant \tfrac{1}{2}\delta\sqrt{n}. \tag{39}$$

The points of the receptive field will be called receptive points.

Here, $\mathrm{Co}(X)$ stands for the convex hull of a finite point set $X = \{x^{(1)}, \ldots, x^{(K)}\} \subset \mathbb{R}^n$:

$$Co(X) = \left\{ \sum_{i=1}^{K} \lambda_i x^{(i)} \,\middle|\, \lambda_i \in \mathbb{R}_{\geqslant 0}, \sum_{i=1}^{K} \lambda_i = 1 \right\}.$$

In Definition 6, condition (35) means that the center of the receptive field belongs to this field. Condition (36) implies that the distance from the central point z to each point g of the receptive field does not exceed $\eta\delta\sqrt{n}$. According to (37), for any two different points $g' \neq g''$ of the receptive field, the distance between them cannot be less than δ. Condition (38) says that for any point g' of the receptive field, there is a point g'' in this field such that the distance between g' and g'' is equal to δ. Condition (39) implies that for any point x belonging to the convex hull of the receptive field, there is a point g in this field such that the distance between x and g does not exceed $\frac{1}{2}\delta\sqrt{n}$. An example of the receptive field in the space $\mathbb{R}^3$ is presented in Fig. 3.

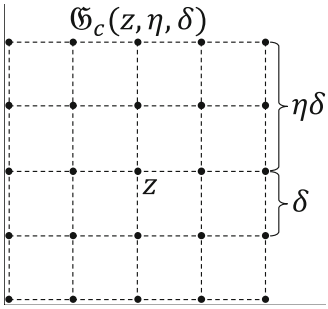

Fig. 3. Receptive field in the space $\mathbb{R}^3$.

Let us describe a constructive method for building a receptive field. Without loss of generality, we assume that $c_n \neq 0$. Consider the following set of vectors:

$$c^{(0)} = c = (c_1, c_2, c_3, c_4, \ldots, c_{n-1}, c_n);$$

$$c^{(1)} = \begin{cases} \left(-\frac{1}{c_1}\sum_{i=2}^{n} c_i^2, c_2, c_3, c_4, \ldots, c_{n-1}, c_n\right), \text{ if } c_1 \neq 0; \\ (1, 0, \ldots, 0), \text{ if } c_1 = 0; \end{cases}$$

$$c^{(2)} = \begin{cases} \left(0, -\frac{1}{c_2}\sum_{i=3}^{n} c_i^2, c_3, c_4, \ldots, c_{n-1}, c_n\right), \text{ if } c_2 \neq 0; \\ (0, 1, 0, \ldots, 0), \text{ if } c_2 = 0; \end{cases}$$

$$c^{(3)} = \begin{cases} \left(0, 0, -\frac{1}{c_3}\sum_{i=4}^{n} c_i^2, c_4, \ldots, c_{n-1}, c_n\right), \text{ if } c_3 \neq 0; \\ (0, 0, 1, 0, \ldots, 0), \text{ if } c_3 = 0; \end{cases}$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$c^{(n-2)} = \begin{cases} \left(0, \ldots, 0, -\frac{1}{c_{n-2}}\sum_{i=n-1}^{n} c_i^2, c_{n-1}, c_n\right), \text{ if } c_{n-2} \neq 0; \\ (0, \ldots, 0, 1, 0, 0), \text{ if } c_{n-2} = 0; \end{cases}$$

$$c^{(n-1)} = \begin{cases} \left(0, \ldots, 0, -\frac{c_n^2}{c_{n-1}}, c_n\right), \text{ if } c_{n-1} \neq 0; \\ (0, \ldots, 0, 0, 1, 0), \text{ if } c_{n-1} = 0. \end{cases}$$

It is easy to see that

$$\forall i, j \in \{0, 1, \ldots, n-1\}, i \neq j : \left\langle c^{(i)}, c^{(j)} \right\rangle = 0.$$

This means that $c_0, \ldots, c_{n-1}$ is an orthogonal basis in $\mathbb{R}^n$. In particular,

$$\forall i = 1, \ldots, n-1 : \left\langle c, c^{(i)} \right\rangle = 0. \tag{40}$$

The following Proposition 4 shows that the linear subspace of the dimension $(n-1)$ generated by the orthogonal vectors $c_1, \ldots, c_{n-1}$ is a hyperplane parallel to the hyperplane H_c.

Proposition 4. *Define the following linear subspace S_c of the dimension $(n-1)$ in $\mathbb{R}^n$:*

$$S_c = \left\{ \sum_{i=1}^{n-1} \lambda_i c^{(i)} \,\middle|\, \lambda_i \in \mathbb{R} \right\}. \tag{41}$$

Then,

$$\forall s \in S_c : s + z \in H_c. \tag{42}$$

Proof. Let $s \in S_c$, i.e.,

$$s = \lambda_1 c^{(1)} + \ldots + \lambda_{n-1} c^{(n-1)}.$$

Then,

$$\langle c, (s+z) - z \rangle = \lambda_1 \left\langle c, c^{(1)} \right\rangle + \ldots + \lambda_{n-1} \left\langle c, c^{(n-1)} \right\rangle.$$

In view of (40), this implies

$$\langle c, (s+z) - z \rangle = 0.$$

Comparing this with (16), we obtain $s + z \in H_c$. □

Define the following set of vectors:

$$e^{(i)} = \frac{c^{(i)}}{\|c^{(i)}\|} \; (i = 1, \ldots, n-1). \tag{43}$$

It is easy to see that the set $\{e_1, \ldots, e_{n-1}\}$ is an orthonormal basis of the subspace S_c.

The procedure for constructing a receptive field is presented as Algorithm 1. This algorithm constructs a receptive field $\mathfrak{G}(z, \eta, \delta)$ consisting of

$$K_{\mathfrak{G}} = (2\eta + 1)^{n-1} \tag{44}$$

points. These points are arranged at the nodes of a regular lattice having the form of a hypersquare (a hypercube of the dimension $n-1$) with the edge length equal to $2\eta\delta$. The edge length of the unit cell is δ. According to Step 13 of Algorithm 1 and Proposition 4, this hypersquare lies in the hyperplane H_c and has the center at the point z. The drawback of Algorithm 1 is that the number of nested **for** loops depends on the dimension of the space. This issue can be solved using the function G, which calculates a point of the receptive field by its ordinal number (numbering starts from zero; the order is determined by Algorithm 1). The implementation of the function G is represented as Algorithm 2. The following Proposition 5 provides an estimation of the time complexity of Algorithm 2.

Proposition 5. *Algorithm 2 enables an implementation that has time complexity*[1]

$$c_G = 4n^2 + 5n - 9, \tag{45}$$

where n is the space dimension.

[1] Here, time complexity refers to the number of arithmetic and comparison operations required to execute the algorithm.

Algorithm 1. Building a receptive field $\mathfrak{G}(z, \eta, \delta)$

Require: $z \in H_c$, $\eta \in \mathbb{N}$, $\delta \in \mathbb{R}_{>0}$
1: $\mathfrak{G} := \emptyset$
2: **for** $i_{n-1} = 0 \ldots 2\eta$ **do**
3: $\quad s_{n-1} := i_{n-1}\delta - \eta\delta$
4: $\quad$ **for** $i_{n-2} = 0 \ldots 2\eta$ **do**
5: $\quad\quad s_{n-2} := i_{n-2}\delta - \eta\delta$
6: $\quad\quad \ldots$
7: $\quad\quad$ **for** $i_1 = 0 \ldots 2\eta$ **do**
8: $\quad\quad\quad s_1 := i_1\delta - \eta\delta$
9: $\quad\quad\quad s := \mathbf{0}$
10: $\quad\quad\quad$ **for** $j = 1 \ldots n-1$ **do**
11: $\quad\quad\quad\quad s := s + s_j e^{(j)}$
12: $\quad\quad\quad$ **end for**
13: $\quad\quad\quad \mathfrak{G} := \mathfrak{G} \cup \{s + z\}$
14: $\quad\quad$ **end for**
15: $\quad$ **end for**
16: **end for**

Proof. Consider Algorithm 3 representing a low-level implementation of Algorithm 2. The values calculated in Steps 1–2 of Algorithm 3 do not depend on the receptive point number k and therefore can be considered constants. In Steps 3–8, the **repeat/until** loop runs $(n-1)$ times and requires $c_{3:8} = 5(n-1)$ operations. In steps 13–16, the nested **repeat/until** loop runs n times and requires $c_{13:16} = 4n$ operations. In steps 10–18, the external **repeat/until** loop runs $(n-1)$ times and requires $c_{10:18} = (4 + c_{13-16})(n-1) = 4(n^2-1)$ operations. In total, we obtain

$$c_G = c_{3:8} + c_{10:18} = 4n^2 + 5n - 9.$$

□

Corollary 1. *The time complexity of Algorithm 2 can be estimated as* $O(n^2)$.

Definition 7. *Let* $z \in H_c$. *Fix* $\eta \in \mathbb{N}$, $\delta \in \mathbb{R}_{>0}$. *The image* $\mathfrak{I}(z, \eta, \delta)$ *generated by the receptive field* $\mathfrak{G}(z, \eta, \delta)$ *is an ordered set of real numbers defined by the equation*

$$\mathfrak{I}(z, \eta, \delta) = \{\rho_c(\gamma_M(g)) |\, g \in \mathfrak{G}(z, \eta, \delta)\}. \tag{46}$$

The order of the real numbers in the image is determined by the order of the respective receptive points.

Algorithm 2. The function G calculates a receptive point by its number k

Require: $z \in H_c$, $\eta \in \mathbb{N}$, $\delta \in \mathbb{R}_{>0}$
function $\mathrm{G}(k, n, z, \eta, \delta)$
  for $j = (n-1) \ldots 1$ **do**
    $i_j := \lfloor k/(2\eta+1)^{j-1} \rfloor$
    $k := k \mod (2\eta+1)^{j-1}$
  end for
  $g := z$
  for $j = 1 \ldots (n-1)$ **do**
    $g := g + (i_j\delta - \eta\delta)e^{(j)}$
  end for
  $\mathrm{G} := g$
end function

The following Algorithm 4 implements the function $\mathfrak{I}(z, \eta, \delta)$ building an image as a list of real numbers.

Algorithm 3. Low-level implementation of Algorithm 2

$p := 2\eta + 1$; $r := \eta\delta$; $h := p^{n-2}$; $g := z$
$j := n - 1$
repeat
  $l_j := \lfloor k/h \rfloor$
  $k := k \mod h$
  $h := h/p$
  $j := j - 1$
until $j = 0$
$j := 1$
repeat
  $w_j := l_j\delta - r$
  $i := 1$
  repeat
    $g_i := g_i + w_j e_i^{(j)}$
    $i := i + 1$
  until $i > n$
  $j := j + 1$
until $j = n$

Here, [] stands for the empty list, and $+\!\!+$ stands for the operation of list concatenation.

Let $\langle \tilde{a}_i, c \rangle > 0$. This means that the half-space H_i^+ is recessive with respect to the vector c (see Proposition 1). Let there be a point $u \in H_i \cap M$. Assume

that we managed to create an artificial neural network DNN, which receives the image $\mathfrak{I}(\pi_c(u), \eta, \delta)$ as an input and outputs the point u' such that

$$u' = \arg\min \{\rho_c(x) | \, x \in H_i \cap M\}.$$

Then, we can build the following Algorithm 5 solving linear programming problem (20) using the DNN.

Algorithm 4. Building an image $\mathfrak{I}(z, \eta, \delta)$

Require: $z \in H_c$, $\eta \in \mathbb{N}$, $\delta \in \mathbb{R}_{>0}$

function $\mathfrak{I}(z, \eta, \delta)$
  $\mathfrak{I} := [\,]$
  for $k = 0 \ldots ((2\eta+1)^{n-1} - 1)$ **do**
    $g_k := \mathrm{G}(k, n, z, \eta, \delta)$
    $\mathfrak{I} := \mathfrak{I} +\!\!+ [\rho_c(\gamma_M(g_k))]$
  end for
end function

Algorithm 5. Linear programming using a DNN

Require: $u^{(1)} \in H_i \cap M$, $\langle \tilde{a}_i, c \rangle > 0$, $z \in H_c$; $\eta \in \mathbb{N}$, $\delta \in \mathbb{R}_{>0}$

$k := 1$
repeat
  $\mathcal{I} := \mathfrak{I}(u^{(k)}, \eta, \delta)$
  $u^{(k+1)} := \mathrm{DNN}(\mathcal{I})$
  $k := k + 1$
until $u^{(k)} \neq u^{(k-1)}$
$\bar{x} := u^{(k)}$
stop

Only an outline of the forthcoming algorithm is presented here, it needs further formalization, detalization and refinement.

3 Parallel Algorithm for Building an LP Problem Image

When solving LP problems of large dimension with a large number of constraints, Algorithm 4 of building an LP problem image can incur significant runtime overhead. This section presents a parallel version of Algorithm 4, which significantly reduces the runtime overhead of building the image of a large-scale LP problem. The parallel implementation of Algorithm 4 is based on the BSF parallel computation model [27,28]. The BSF model is intended for a cluster computing system, uses the master/worker paradigm and requires the representation of the algorithm in the form of operations on lists using higher-order functions *Map* and *Reduce* defined in the Bird–Meertens formalism [1]. The BSF model also

provides a cost metric for the analytical evaluation of the scalability of a parallel algorithm that meets the specified requirements. Examples of the BSF model application can be found in [7,30–33].

Let us represent Algorithm 4 in the form of operations on lists using higher-order functions *Map* and *Reduce*. We use the list of ordinal numbers of inequalities of system (4) as a list, which is the second parameter of the higher-order function *Map*:

$$\mathcal{L}_{map} = [1, \ldots, m]\,. \tag{47}$$

Designate $\mathbb{R}_\infty = \mathbb{R} \cup \{\infty\}$. We define a parameterized function

$$\mathrm{F}_k : \{1, \ldots, m\} \to \mathbb{R}_\infty,$$

which is the first parameter of the higher-order function *Map*, as follows:

$$\mathrm{F}_k(i) = \begin{cases} \rho_c\left(\gamma_i(g_k)\right), \text{ if } \langle \tilde{a}_i, c\rangle > 0 \text{ and } \gamma_i(g_k) \in M; \\ \infty, \text{ if } \langle \tilde{a}_i, c\rangle \leqslant 0 \text{ or } \gamma_i(g_k) \notin M. \end{cases} \tag{48}$$

where $g_k = G(k, n, z, \eta, \delta)$ (see Algorithm 2), and $\gamma_i(g_k)$ is calculated by Eq. (24). Informally, the function F_k maps the ordinal number of the half-space H_i^+ to the distance from the objective projection to the objective hyperplane if H_i^+ is recessive with respect to c (see Proposition 1), and the objective projection belongs to M. Otherwise, F_k returns the special value ∞.

The higher-order function *Map* transforms the list $\mathcal{L}_{map}$ into the list $\mathcal{L}_{reduce}$ by applying the function F_k to each element of the list $\mathcal{L}_{map}$:

$$\mathcal{L}_{reduce} = \mathrm{Map}\left(\mathrm{F}_k, \mathcal{L}_{map}\right) = [\mathrm{F}_k(1), \ldots, \mathrm{F}_k(m)] = [\rho_1, \ldots, \rho_m]\,.$$

Define the associative binary operation $\Downarrow : \mathbb{R}_\infty \to \mathbb{R}_\infty$ as follows:

$$\begin{gathered} \infty \Downarrow \infty = \infty; \\ \forall \alpha \in \mathbb{R} : \alpha \Downarrow \infty = \alpha; \\ \forall \alpha, \beta \in \mathbb{R} : \alpha \Downarrow \beta = \min(\alpha, \beta). \end{gathered}$$

Informally, the operation $\Downarrow$ calculates the minimum of two numbers.

The higher-order function *Reduce* folds the list $\mathcal{L}_{reduce}$ to the single value $\rho \in \mathbb{R}_\infty$ by sequentially applying the operation $\Downarrow$ to the entire list:

$$\mathrm{Reduce}(\Downarrow, \mathcal{L}_{reduce}) = \rho_1 \Downarrow \rho_2 \Downarrow \ldots \Downarrow \rho_m = \rho.$$

Algorithm 6. Building an image $\mathfrak{I}$ by *Map* and *Reduce*

Require: $z \in H_c$, $\eta \in \mathbb{N}$, $\delta \in \mathbb{R}_{>0}$

input $n, m, A, b, c, z, \eta, \delta$
$\mathfrak{I} := [\,]$
$\mathcal{L}_{map} := [1, \ldots, m]$
for $k = 0 \ldots ((2\eta+1)^{n-1} - 1)$ **do**
  $\mathcal{L}_{reduce} := \mathrm{Map}(\mathrm{F}_k, \mathcal{L}_{map})$
  $\rho := \mathrm{Reduce}(\Cup, \mathcal{L}_{reduce})$
  $\mathfrak{I} := \mathfrak{I} +\!\!+ [\rho]$
end for
output $\mathfrak{I}$
stop

Algorithm 6 builds the image $\mathfrak{I}$ of the LP problem using higher-order functions *Map* and *Reduce*. The parallel version of Algorithm 6 is based on algorithmic template 2 in [28]. The result is presented as Algorithm 7.

Algorithm 7. Parallel algorithm of building the image $\mathfrak{I}$

Master	***l*th Worker (*l*=0,. . . ,*L-1*)**
input n	**input** $n, m, A, b, c, z, \eta, \delta$
$\mathfrak{I} := [\,]$	$L := \mathrm{NumberOfWorkers}$
$k := 0$	$\mathcal{L}_{map(l)} := [lm/L, \ldots, ((l+1)m/L) - 1]$
repeat	**repeat**
SendToWorkers k	**RecvFromMaster** k
	$\mathcal{L}_{reduce(l)} := \mathrm{Map}\left(\mathrm{F}_k, \mathcal{L}_{map(l)}\right)$
	$\rho_l := \mathrm{Reduce}\left(\Cup, \mathcal{L}_{reduce(l)}\right)$
RecvFromWorkers $[\rho_0, \ldots, \rho_{L-1}]$	**SendToMaster** ρ_l
$\rho := \mathrm{Reduce}\,(\Cup, [\rho_0, \ldots, \rho_{L-1}])$	
$\mathfrak{I} := \mathfrak{I} +\!\!+ [\rho]$	
$k := k + 1$	
$exit := \left(k \geqslant (2\eta+1)^{n-1}\right)$	
SendToWorkers $exit$	**RecvFromMaster** $exit$
until $exit$	**until** $exit$
output $\mathfrak{I}$	
stop	**stop**

Let us explain the steps of Algorithm 7. For simplicity, we assume that the number of constraints m is a multiple of the number of workers L. We also assume that the numbering of inequalities starts from zero. The parallel algorithm includes $L + 1$ processes: one master process and L worker processes.

The master manages the computations. In Step 1, the master reads the space dimension n. In Step 2 of the master, the image variable $\mathfrak{I}$ is initialized to the empty list. Step 3 of the master assigns zero to the iteration counter k. At Steps 4–14, the master organizes the **repeat/until** loop, in which the image $\mathfrak{I}$ of the LP problem is built. In Step 5, the master sends the receptive point number g_k to all workers. In Step 8, the master expects particular results from all workers. These particular results are folded to a single value, which is added to the image $\mathfrak{I}$ (Steps 9–10 of the master). Step 11 of the master increases the iteration counter k by 1. Step 12 of the master assigns the logical value $\left(k \geqslant (2\eta+1)^{n-1}\right)$ to the Boolean variable *exit*. In Step 13, the master sends the value of the Boolean variable *exit* to all workers. According to (44), $exit = false$ means that not all the points of the receptive field are processed. In this case, the control is passed to the next iteration of the external **repeat/until** loop (Step 14 of the master). After exiting the **repeat/until** loop, the master outputs the constructed image $\mathfrak{I}$ (Step 15) and terminates its work (Step 16).

All workers execute the same program codes, but with different data. In Step 3, the lth worker defines its own sublist. In Step 4, the worker enters the **repeat/until** loop. In Step 5, it receives the number k of the next receptive point. In Step 6, the worker processes its sublist $\mathcal{L}_{map(l)}$ using the higher-order function *Map*, which applies the parameterized function F_k, defined by (48), to each element of the sublist. The result is the sublist $\mathcal{L}_{reduce(l)}$, which includes the distances $\mathrm{F}_k(i)$ from the objective hyperplane H_c to the objective projections of the receptive point g_k onto the hyperplanes H_i for all i from the sublist $\mathcal{L}_{map(l)}$. In Step 7, the worker uses the higher-order function *Reduce* to fold the sublist $\mathcal{L}_{reduce(l)}$ to the single value of ρ_l, using the associative binary operation $\textcircled{\downarrow}$, which calculates the minimum distance. The computed particular result is sent to the master (Step 8 of the worker). In Step 13, the worker waits for the master to send the value of the Boolean variable *exit*. If the received value is false, the worker continues executing the **repeat/until** loop (Step 14 of the worker). Otherwise, the worker process is terminated in Step 16.

Let us obtain an analytical estimation of the *scalability bound* of parallel Algorithm 7 using the cost metric of the BSF parallel computation model [28]. Here, the scalability bound means the number of workers at which the maximum speedup is achieved. The cost metric of the BSF model includes the following cost parameters for the **repeat/until** loop (Steps 4–14) of parallel Algorithm 7:

m : length of the list $\mathcal{L}_{map}$;
D : latency (time taken by the master to send one byte message to a single worker);
t_c : time taken by the master to send the coordinates of the receptive point to a single worker and receive the computed value from it (including latency);
t_{Map} : time taken by a single worker to process the higher-order function *Map* for the entire list $\mathcal{L}_{map}$;
t_a : time taken by computing the binary operation $\textcircled{\downarrow}$.

According to Eq. (14) from [28], the scalability bound of Algorithm 7 can be estimated as follows:

$$L_{max} = \frac{1}{2}\sqrt{\left(\frac{t_c}{t_a \ln 2}\right)^2 + \frac{t_{Map}}{t_a} + 4m} - \frac{t_c}{t_a \ln 2}. \tag{49}$$

Calculate estimations for the time parameters of Eq. (49). To do this, we introduce the following notation for a single iteration of the **repeat/until** loop (Steps 4–14) of Algorithm 7:

c_c : quantity of numbers sent from the master to the worker and back within one iteration;
c_{Map} : quantity of arithmetic and comparison operations computed in Step 5 of serial algorithm 6;
c_a : quantity of arithmetic and comparison operations required to compute the binary operation $\textcircled{\downarrow}$.

At the beginning of every iteration, the master sends each worker the receptive point number k. In response, the worker sends the distance from the receptive point g_k to its objective projection. Therefore,

$$c_c = 2. \tag{50}$$

In the context of Algorithm 6

$$c_{Map} = (c_G + c_{F_k})\, m, \tag{51}$$

where c_G is the number of operations taken to compute the coordinates of the point g_k, and c_{F_k} is the number of operations required to calculate the value of $\mathrm{F}_k(i)$, assuming that the coordinates of the point g_k have already been calculated. The estimation of c_G is provided by Proposition 5. Let us estimate c_{F_k}. According to (24), calculating the objective projection $\gamma_i(g)$ takes $(6n-2)$ arithmetic operations. It follows from (19) that the calculation of $\rho_c(x)$ takes $(5n-1)$ arithmetic operations. Inequalities (4) imply that checking the condition $x \in M$ takes $m(2n-1)$ arithmetic operations and m comparison operations. Hence, $\mathrm{F}_k(i)$ takes a total of $(2mn + 11n - 3)$ operations. Thus,

$$c_{F_k} = 2mn + 11n - 3. \tag{52}$$

Substituting the right-hand sides of Eqs. (45) and (52) in (51), we obtain

$$c_{Map} = 4n^2m + 2m^2n + 16nm - 12m. \tag{53}$$

To perform the binary operation $\textcircled{\downarrow}$, one comparison operation must be executed:

$$c_a = 1. \tag{54}$$

Let τ_{op} stand for the average execution time of arithmetic and comparison operations, and let τ_{tr} stand for the average time of sending a single real number (excluding latency). Then, using Eqs. (50), (53), and (54) we obtain

$$t_c = c_c\tau_{tr} + 2D = 2(\tau_{tr} + D); \tag{55}$$

$$t_{Map} = c_{Map}\tau_{op} = (4n^2m + 2m^2n + 16nm - 12m)\tau_{op}; \tag{56}$$

$$t_a = c_a\tau_{op} = \tau_{op}. \tag{57}$$

Substituting the right-hand sides of Eqs. (55)–(57) in (49), we obtain the following estimations of the scalability bound of Algorithm 7:

$$L_{max} = \frac{1}{2}\sqrt{\left(\frac{2(\tau_{tr} + D)}{\tau_{op}\ln 2}\right)^2 + 4n^2m + 2m^2n + 16nm - 12m} - \frac{2(\tau_{tr} + D)}{\tau_{op}\ln 2}.$$

where n is the space dimension, m is the number of constraints, D is the latency. For large values of m and n, this is equivalent to

$$L_{max} \approx O(\sqrt{2n^2m + m^2n + 8nm - 6m}). \tag{58}$$

If we assume that $m = O(n)$, then it follows from (58) that

$$L_{max} \approx O(n\sqrt{n}), \tag{59}$$

where n is the space dimension. Estimation (59) allows us to conclude that Algorithm 7 scales very well[2]. In the following section, we verify analytical estimation (59) by conducting large-scale computational experiments on a real cluster computing system.

4 Computational Experiments

We performed a parallel implementation of Algorithm 7 in the form of the ViLiPP (Visualization of Linear Programming Problem) program in C++ using a BSF-skeleton [29]. The BSF-skeleton based on the BSF parallel computation model encapsulates all aspects related to the parallelization of the program using the MPI [9] library and the OpenMP [13] programming interface. The source code of the ViLiPP program is freely available on the Internet at https://github.com/nikolay-olkhovsky/LP-visualization-MPI. Using the ViLiPP parallel program, we conducted experiments to evaluate the scalability of Algorithm 7 on the "Tornado SUSU" cluster computing system [16], the characteristics of which are presented in Table 1.

To conduct computational experiments, we constructed three random LP problems using the FRaGenLP problem generator [32]. The parameters of these

[2] Let $L_{max} = O(n^\alpha)$. We say: the algorithm *scales perfectly* if $\alpha > 1$; the algorithm *scales well* if $\alpha = 1$; the algorithm demonstrates *limited scalability* if $0 < \alpha < 1$; the algorithm does *not scale* if $\alpha = 0$.

Table 1. Specifications of the "Tornado SUSU" computing cluster

Parameter	Value
Number of processor nodes	480
Processor	Intel Xeon X5680 (6 cores, 3.33 GHz)
Processors per node	2
Memory per node	24 GB DDR3
Interconnect	InfiniBand QDR (40 Gbit/s)
Operating system	Linux CentOS

Table 2. Parameters of test LP problems

Problem ID	Dimension	Number of constraints	Non-zero values in A	Receptive field cardinality
$LP7$	7	4 016	100%	15 625
$LP6$	6	4 014	100%	3 125
$LP5$	5	4 012	100%	625

problems are given in Table 2. In all cases, the number of non-zero values of the matrix A of problem (1) was 100%. For all problems, the rank η of the receptive field was assumed to be equal to 2. In accordance with Eq. (44), the receptive field cardinality demonstrated an exponential growth with an increase in the space dimension.

The results of the computational experiments are presented in Table 3 and in Fig. 4. In all runs, a separate processor node was allocated for each worker. One more separate processor node was allocated for the master. The computational experiments show that the ViLiPP program scalability bound increases with an increase in the problem dimension. For LP5, the maximum of the speedup curve is reached around 190 nodes. For LP6, the maximum is located around 260 nodes. For LP7, the scalability bound is approximately equal to 326 nodes. At the same time, there is an exponential increase in the runtime of building the LP problem image. Building the LP5 problem image takes 10 s on 11 processor nodes. Building the LP7 problem image takes 5 min on the same number of nodes. An additional computational experiment shows that building an image of the problem with $n = 9$ takes 1.5 h on 11 processor nodes.

The conducted experiments show that on the current development level of high-performance computing, the proposed method is applicable to solving LP problems that include up to 100 variables and up to 100 000 constraints.

Table 3. Runtime of building an LP problem image (sec.)

Number of processor nodes	LP5	LP6	LP7
11	9.81	54.45	303.78
56	1.93	10.02	59.43
101	1.55	6.29	33.82
146	1.39	4.84	24.73
191	1.35	4.20	21.10
236	1.38	3.98	19.20
281	1.45	3.98	18.47
326	1.55	4.14	18.30

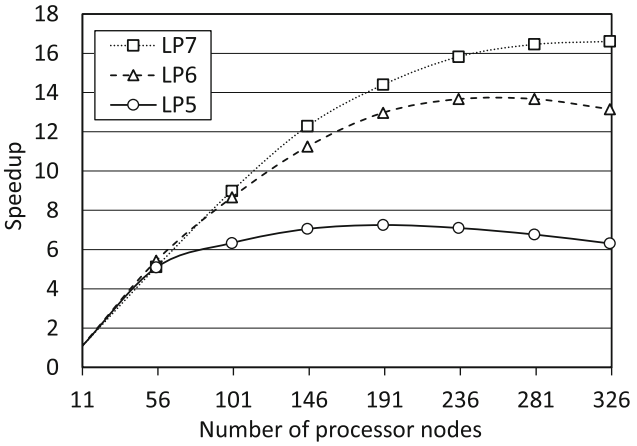

Fig. 4. ViLiPP parallel program speedup for LP problems of various sizes.

5 Conclusion

The main contribution of this work is a mathematical model of the visual representation of a multidimensional linear programming problem of finding the maximum of a linear objective function in a feasible region. The central element of the model is the receptive field, which is a finite set of points located at the nodes of a square lattice constructed inside a hypercube. All points of the receptive field lie in the objective hyperplane orthogonal to the vector $c = (c_1, \ldots, c_n)$, which is composed of the coefficients of the linear objective function. The target hyperplane is placed so that for any point x from the feasible region and any point z of the objective hyperplane, the inequality $\langle c, x\rangle < \langle c, z\rangle$ holds. We can say that the receptive field is a multidimensional abstraction of the digital camera image sensor. From each point of the receptive field, we construct a ray parallel to the vector c and directed to the side of the feasible region. The point

at which the ray hits the feasible region is called the objective projection. The image of the linear programming problem is a matrix of the dimension $(n-1)$, in which each element is the distance from the point of the receptive field to the corresponding point of the objective projection.

The algorithm for calculating the coordinates of a receptive field point by its ordinal number is described. It is shown that the time complexity of this algorithm can be estimated as $O(n^2)$, where n is the space dimension. An outline of the algorithm for solving the linear programming problem by an artificial neural network using the constructed images is presented. A parallel algorithm for constructing the image of a linear programming problem on computing clusters is proposed. This algorithm is based on the BSF parallel computation model, which uses the master/workers paradigm and assumes a representation of the algorithm in the form of operations on lists using higher-order functions *Map* and *Reduce*. It is shown that the scalability bound of the parallel algorithm admits the estimation of $O(n\sqrt{n})$. This means that the algorithm demonstrates good scalability.

The parallel algorithm for constructing the multidimensional image of a linear programming problem is implemented in C++ using the BSF–skeleton that encapsulates all aspects related to parallelization by the MPI library and the OpenMP API. Using this software implementation, we conducted large-scale computational experiments on constructing images for random multidimensional linear programming problems with a large number of constraints on the "Tornado SUSU" computing cluster. The conducted experiments confirm the validity and efficiency of the proposed approaches. At the same time, it should be noted that the time of image construction increases exponentially with an increase in the space dimension. Therefore, the proposed method is applicable to problems with the number of variables not exceeding 100. However, the number of constraints can theoretically be unbounded.

Future research directions are as follows.

1. Develop a method for solving linear programming problems based on the analysis of their images and prove its convergence.
2. Develop and implement a method for training data set generation to create a neural network that solves linear programming problems by analyzing their images.
3. Develop and train an artificial neural network solving multidimensional linear programming problems.
4. Develop and implement a parallel program on a computing cluster that constructs multidimensional images of a linear programming problem and calculates its solution using an artificial neural network.

Funding Information. The study was partially funded by the Russian Foundation for Basic Research (project No. 20-07-00092-a) and the Ministry of Science and Higher Education of the Russian Federation (government order FENU-2020-0022).

References

1. Bird, R.S.: Lectures on constructive functional programming. In: Broy, M. (ed.) Constructive Methods in Computing Science. NATO ASI Series, vol. 55, pp. 151–216. Springer, Heidelberg (1988). https://doi.org/10.1007/978-3-642-74884-4_5
2. Bixby, R.: Solving real-world linear programs: a decade and more of progress. Oper. Res. **50**(1), 3–15 (2002). https://doi.org/10.1287/opre.50.1.3.17780
3. Brogaard, J., Hendershott, T., Riordan, R.: High-frequency trading and price discovery. Rev. Financ. Stud. **27**(8), 2267–2306 (2014). https://doi.org/10.1093/rfs/hhu032
4. Chung, W.: Applying large-scale linear programming in business analytics. In: 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), pp. 1860–1864. IEEE (2015). https://doi.org/10.1109/IEEM.2015.7385970
5. Dantzig, G.: Linear Programming and Extensions. Princeton University Press, Princeton (1998)
6. Dongarra, J., Gottlieb, S., Kramer, W.: Race to exascale. Comput. Sci. Eng. **21**(1), 4–5 (2019). https://doi.org/10.1109/MCSE.2018.2882574
7. Ezhova, N.A., Sokolinsky, L.B.: Scalability evaluation of iterative algorithms used for supercomputer simulation of physical processes. In: Proceedings - 2018 Global Smart Industry Conference, GloSIC 2018, Art. No. 8570131, p. 10. IEEE (2018). https://doi.org/10.1109/GloSIC.2018.8570131
8. Gondzio, J., Gruca, J.A., Hall, J., Laskowski, W., Zukowski, M.: Solving large-scale optimization problems related to Bell's Theorem. J. Comput. Appl. Math. **263**, 392–404 (2014). https://doi.org/10.1016/j.cam.2013.12.003
9. Gropp, W.: MPI 3 and beyond: why MPI is successful and what challenges it faces. In: Träff, J.L., Benkner, S., Dongarra, J.J. (eds.) EuroMPI 2012. LNCS, vol. 7490, pp. 1–9. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33518-1_1
10. Hafsteinsson, H., Levkovitz, R., Mitra, G.: Solving large scale linear programming problems using an interior point method on a massively parallel SIMD computer. Parallel Algorithms Appl. **4**(3–4), 301–316 (1994). https://doi.org/10.1080/10637199408915470
11. Hartung, T.: Making big sense from big data. Front. Big Data **1**, 5 (2018). https://doi.org/10.3389/fdata.2018.00005
12. Jagadish, H.V., et al.: Big data and its technical challenges. Commun. ACM **57**(7), 86–94 (2014). https://doi.org/10.1145/2611567
13. Kale, V.: Shared-memory parallel programming with OpenMP. In: Parallel Computing Architectures and APIs, chap. 14, pp. 213–222. Chapman and Hall/CRC, Boca Raton (2019). https://doi.org/10.1201/9781351029223-18/SHARED-MEMORY-PARALLEL-PROGRAMMING-OPENMP-VIVEK-KALE
14. Karmarkar, N.: A new polynomial-time algorithm for linear programming. Combinatorica **4**(4), 373–395 (1984). https://doi.org/10.1007/BF02579150
15. Karypis, G., Gupta, A., Kumar, V.: A parallel formulation of interior point algorithms. In: Proceedings of the 1994 ACM/IEEE Conference on Supercomputing (Supercomputing 1994), Los Alamitos, CA, USA, pp. 204–213. IEEE Computer Society Press (1994). https://doi.org/10.1109/SUPERC.1994.344280
16. Kostenetskiy, P., Semenikhina, P.: SUSU supercomputer resources for industry and fundamental science. In: Proceedings - 2018 Global Smart Industry Conference, GloSIC 2018, Art. No. 8570068, p. 7. IEEE (2018). https://doi.org/10.1109/GloSIC.2018.8570068

17. Lachhwani, K.: Application of neural network models for mathematical programming problems: a state of art review. Arch. Comput. Methods Eng. **27**(1), 171–182 (2019). https://doi.org/10.1007/s11831-018-09309-5
18. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015). https://doi.org/10.1038/nature14539
19. Mamalis, B., Pantziou, G.: Advances in the parallelization of the simplex method. In: Zaroliagis, C., Pantziou, G., Kontogiannis, S. (eds.) Algorithms, Probability, Networks, and Games. LNCS, vol. 9295, pp. 281–307. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24024-4_17
20. Prieto, A., et al.: Neural networks: an overview of early research, current frameworks and new challenges. Neurocomputing **214**, 242–268 (2016). https://doi.org/10.1016/j.neucom.2016.06.014
21. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015). https://doi.org/10.1016/j.neunet.2014.09.003
22. Sodhi, M.: LP modeling for asset-liability management: a survey of choices and simplifications. Oper. Res. **53**(2), 181–196 (2005). https://doi.org/10.1287/opre.1040.0185
23. Sokolinskaya, I.: Parallel method of pseudoprojection for linear inequalities. In: Sokolinsky, L., Zymbler, M. (eds.) PCT 2018. CCIS, vol. 910, pp. 216–231. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99673-8_16
24. Sokolinskaya, I., Sokolinsky, L.B.: On the solution of linear programming problems in the age of big data. In: Sokolinsky, L., Zymbler, M. (eds.) PCT 2017. CCIS, vol. 753, pp. 86–100. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67035-5_7
25. Sokolinskaya, I., Sokolinsky, L.B.: Scalability evaluation of NSLP algorithm for solving non-stationary linear programming problems on cluster computing systems. In: Voevodin, V., Sobolev, S. (eds.) RuSCDays 2017. Communications in Computer and Information Science, vol. 793, pp. 40–53. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71255-0_4
26. Sokolinskaya, I.M., Sokolinsky, L.B.: Scalability evaluation of cimmino algorithm for solving linear inequality systems on multiprocessors with distributed memory. Supercomput. Front. Innov. **5**(2), 11–22 (2018). https://doi.org/10.14529/jsfi180202
27. Sokolinsky, L.B.: Analytical estimation of the scalability of iterative numerical algorithms on distributed memory multiprocessors. Lobachevskii J. Math. **39**(4), 571–575 (2018). https://doi.org/10.1134/S1995080218040121
28. Sokolinsky, L.B.: BSF: a parallel computation model for scalability estimation of iterative numerical algorithms on cluster computing systems. J. Parallel Distrib. Comput. **149**, 193–206 (2021). https://doi.org/10.1016/j.jpdc.2020.12.009
29. Sokolinsky, L.B.: BSF-skeleton: a template for parallelization of iterative numerical algorithms on cluster computing systems. MethodsX **8**, Article Number 101,437 (2021). https://doi.org/10.1016/j.mex.2021.101437
30. Sokolinsky, L.B., Sokolinskaya, I.M.: Scalable method for linear optimization of industrial processes. In: Proceedings - 2020 Global Smart Industry Conference, GloSIC 2020, pp. 20–26. Article Number 9267,854. IEEE (2020). https://doi.org/10.1109/GloSIC50886.2020.9267854
31. Sokolinsky, L.B., Sokolinskaya, I.M.: Scalable parallel algorithm for solving non-stationary systems of linear inequalities. Lobachevskii J. Math. **41**(8), 1571–1580 (2020). https://doi.org/10.1134/S1995080220080181

32. Sokolinsky, L.B., Sokolinskaya, I.M.: FRaGenLP: a generator of random linear programming problems for cluster computing systems. In: Sokolinsky, L., Zymbler, M. (eds.) PCT 2021. CCIS, vol. 1437, pp. 164–177. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81691-9_12
33. Sokolinsky, L.B., Sokolinskaya, I.M.: VaLiPro: linear programming validator for cluster computing systems. Supercomput. Front. Innov. **8**(3), 51–61 (2021). https://doi.org/10.14529/js210303
34. Tolla, P.: A survey of some linear programming methods. In: Paschos, V.T. (ed.) Concepts of Combinatorial Optimization, 2 edn, chap. 7, pp. 157–188. Wiley, Hoboken (2014). https://doi.org/10.1002/9781119005216.ch7
35. Zadeh, N.: A bad network problem for the simplex method and other minimum cost flow algorithms. Math. Program. **5**(1), 255–266 (1973). https://doi.org/10.1007/BF01580132