Article

Apex Method: A New Scalable Iterative Method for Linear Programming

Leonid B. Sokolinsky[1,†,‡,*] and Irina M. Sokolinskaya[2,†,‡]

[1] South Ural State University (National Research University); leonid.sokolinsky@susu.ru
[2] South Ural State University (National Research University); irina.sokolinskaya@susu.ru
* Correspondence: leonid.sokolinsky@susu.ru
† Current address: 76, Lenin prospekt, Chelyabinsk, Russia, 454080
‡ These authors contributed equally to this work.

Abstract: The article presents a new scalable iterative method for linear programming called the "apex method". The key feature of this method is constructing a path close to optimal on the surface of the feasible region from a certain starting point to the exact solution of linear programming problem. The optimal path refers to a path of minimum length according to the Euclidean metric. The apex method is based on the predictor-corrector framework and proceeds in two stages: quest (predictor) and target (corrector). The quest stage calculates a rough initial approximation of linear programming problem. The target stage refines the initial approximation with a given precision. The main operation used in the apex method is an operation that calculates the pseudoprojection, which is a generalization of the metric projection to a convex closed set. This operation is used both in the quest stage and in the target stage. A parallel algorithm using a Fejér mapping to compute the pseudoprojection is presented. An analytical estimation of the parallelism degree of this algorithm is obtained. Also, an algorithm implementing the target stage is given. The convergence of this algorithm is proven. An experimental study of the scalability of the apex method on a cluster computing system is described. The results of applying the apex method to solve problems from the Netlib-LP repository are presented.

Keywords: linear programming; apex method; iterative method; projection-type method; Fejér mapping; parallel algorithm; cluster computing system; scalability evaluation; Netlib-LP repository

1. Introduction

This article is an expanded and revised version of the conference paper [1]. The following motivation encouraged us to delve into this subject. The rapid development of big data storage and processing technologies [2,3] has led to the emergence of optimization mathematical models in the form of multidimensional linear programming problems [4]. Such problems arise in industry, economics, logistics, statistics, quantum physics and other fields [5–7]. An important class of LP applications are non-stationary problems related to optimization in dynamic environments [8]. For a non-stationary LP problem, the objective function and/or constraints change over the computational process. Examples of non-stationary problems are the following: decision support in high-frequency trading [9,10], hydro-gas-dynamics problems [11], optimal control of technological processes [12–14], transportation [15–17], scheduling [18,19].

One of the standard approaches to solving non-stationary optimization problems is to consider each change as the appearance of a new optimization problem that needs to be solved from scratch [8]. However, this approach is often impractical because solving a problem from scratch without reusing information from the past can take too much time. Thus, it is desirable to have an optimization algorithm capable of continuously adapting the solution to the changing environment, reusing information obtained in the past. This approach is applicable for real-time processes if the algorithm tracks the trajectory of the moving optimal point fast enough. In the case of large-scale LP problems, the last

 doi:10.20944/preprints202303.0297.v1

requirement necessitates the development of scalable methods and parallel algorithms of LP.

Until now, one of the most common ways to solve the LP problem was a class of algorithms proposed and developed by Danzig based on the simplex method [20]. The simplex method was found to be efficient for solving a large class of LP problems. In particular, the simplex method easily takes advantage of any hyper-sparsity in a LP problem [21]. However, the simplex method has some fundamental features that limit its use for the solution of large LP problems. First, in certain cases, the simplex method has to iterate through all the vertices of the simplex, which corresponds to exponential time complexity [22–24]. 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 [25]. Such a loss of precision cannot be compensated even by applying such computational intensive procedures as "affine scaling" or "iterative refinement" [26]. Third, in the general case, the sequential nature of simplex method makes it difficult to parallelize on multiprocessor systems with distributed memory [27]. Numerous attempts have been made to make a scalable parallel implementation of the simplex method, but all of them were unsuccessful [28]. In all cases, the scalability boundary was from 16 to 32 processor nodes (see, for example, [29]). Khachiyan proved [30], using a variant of the ellipsoid method (proposed in the 1970's by Shor [31] and Yudin and Nemirovskii [32]), that LP problems can be solved in polynomial time. However, attempts to apply this approach in practice were unsuccessful, since in the vast majority of cases, the ellipsoid method demonstrated much worse efficiency compared to the simplex method. Later, Karmarkar [33] proposed a polynomial-time interior-point algorithm, which can be used in practice. This algorithm generated the whole field of modern interior-point methods [34], which are able to solve large-scale LP problems with millions of variables and millions of equations [35–39]. Moreover, these methods are self-correcting, and therefore provide high precision of calculations. A general lack of interior-point methods is the need to find some feasible point satisfying all the constraints of the LP problem before starting calculations. Finding such an interior point can be reduced to solving an additional LP problem [40]. Another method for finding an interior point is the pseudo-projection method [41], which uses Fejér mappings [42]. One more significant disadvantage of the interior-point method is its poor scalability on multiprocessor systems with distributed memory. There are several successful parallel implementations of the interior-point method for particular cases (see, for example, [43]), but, in the general case, an efficient parallel implementation on multiprocessor systems could not be built. In accordance with this, the development and research of new approaches to solving multidimensional non-stationary LP problems in real time is an urgent direction.

One of the most promising approaches to solving complex problems in real time is the use of neural network models [44]. Artificial neural networks (ANN) are a powerful universal tool that is applicable to solving problems in almost all areas. The most popular neural network model is the feedforward neural network. Training and operation of such networks can be implemented very efficiently on GPUs [45]. An important property of a feedforward neural network is that the time to solve a problem is a known constant that does not depend on the problem parameters. This feature is necessary for real-time mode. Pioneering work on the use of neural networks to solve LP problems is the article by Tank and Hopfield [46]. The article describes a two-layer recurrent neural network. The number of neurons in the first layer is the number of variables of the LP problem. The number of neurons in the second layer coincides with the number of constraints of the LP problem. The first and second layers are fully connected. The weights and biases are uniquely determined by the coefficients and the right-hand sides of the linear inequalities defining the constraints, and the coefficients of the linear objective function. Thus, this network does not require training. The state of the neural network is described by the differential equation $\dot{x}(t) = \nabla E(x(t))$, where $E(x(t))$ is an energy function of a special type. Initially, an arbitrary point of the feasible region is fed to the input of the neural

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

network. Then, the signal of the second layer is recursively fed to the first layer. Such process leads to convergence to a stable state in which the output remains constant. This state corresponds to the minimum of the energy function, and the output signal is a solution to the LP problem. The Tank and Hopfield approach has been expanded and improved in numerous works (see, for example, [47–51]). The main disadvantage of this approach is the unpredictable number of work cycles of the neural network. Therefore, a recurrent network based on an energy function cannot be used to solve large-scale LP problems in real time.

In the recent paper [52], a n-dimensional mathematical model for visualizing LP problems was proposed. This model makes it possible to use feedforward neural networks, including convolutional networks [53], to solve multidimensional LP problems, the feasible region of which is a bounded non-empty set. However, there are practically no works in scientific periodicals devoted to the use of convolutional neural networks for solving LP problems [54]. The reason is that convolutional neural networks focus on image processing, but there are no methods for constructing training datasets based on a visual representation of the multidimensional LP problems.

This article describes a new scalable iterative method for solving multidimensional LP problems. This method called "apex method". The apex method allows you to generate training datasets for the development of feedforward neural networks capable of finding a solution to a multidimensional LP problem based on its visual representation. The apex method is based on the predictor-corrector framework. At the prediction step, a point belonging to the feasible region of the LP problem is calculated. The corrector step calculates a sequence of points converging to the exact solution of the LP problem. The rest of the paper is organized as follows. Section 2 provides a review of iterative projection-type methods and algorithms for solving linear feasibility problems and LP problems. Section 3 includes a theoretical basis of the apex method. Section 4 presents a formal description of apex method. Section 4.1 considers the implementation of the pseudoprojection operation in the form of sequential and parallel algorithms and provides an analytical estimation of the scalability of parallel implementation. Section 4.2 describes the quest stage. Section 4.3 describes the target stage. Section 5 presents an information about the software implementation of the apex method and describes the results of large-scale computational experiments on a cluster computing system. Section 6 discusses the issues related to the main contribution of this article, the advantages and disadvantages of the proposed approach, possible applications and some other aspects of using the apex method. Finally, in Section 7, we present our conclusions and comment on possible further studies of the apex method.

2. Related Work

This section provides an overview of works devoted to iterative projection-type methods used to solve linear feasibility and LP problems. The linear feasibility problem can be stated as follows. Consider the system of linear inequalities in matrix form

$$Ax \leqslant b, \tag{1}$$

where $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^n$. To avoid triviality, we assume that $m > 1$. The linear feasibility problem consists of finding a point $\tilde{x} \in \mathbb{R}^n$ satisfying matrix inequality system (1). We assume from now on that such a point exists.

Projection-type methods rely on the following geometric interpretation of the linear feasibility problem. Let $a_i \in \mathbb{R}^n$ 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

$$\langle a_i, x \rangle \leqslant b_i, i = 1, \ldots, m. \tag{2}$$

Here and further on, $\langle \cdot, \cdot \rangle$ stands for the dot product of vectors. We assume from now on that

$$a_i \neq \mathbf{0} \tag{3}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

for all $i = 1, \ldots, m$. For each inequality $\langle a_i, x \rangle \leqslant b_i$, define the closed half space

$$\hat{H}_i = \{x \in \mathbb{R}^n | \langle a_i, x \rangle \leqslant b_i\}, \tag{4}$$

and its bounding hyperplane

$$H_i = \{x \in \mathbb{R}^n | \langle a_i, x \rangle = b_i\}. \tag{5}$$

For any point $x \in \mathbb{R}^n$, the *orthogonal projection* $\pi(x)$ of point x onto the hyperplane H_i can be calculated by the equation

$$\pi_i(x) = x - \frac{\langle a_i, x \rangle - b_i}{\|a_i\|^2} a_i. \tag{6}$$

Here and below, $\| \cdot \|$ denotes the Euclidean norm. Let us define

$$M = \bigcap_{i=1}^{m} \hat{H}_i. \tag{7}$$

We assume that $M \neq \emptyset$, i.e. the solution of system (1) exists. In this case, M is a convex closed polytope in the space $\mathbb{R}^n$. In geometric interpretation, the linear feasibility problem consists of finding a point $\tilde{x} \in M$.

The forefathers of the iterative projection-type methods for solving linear feasibility problems are Kaczmarz and Cimmino. In [55] (English translation in [56]), Kaczmarz presented a sequential projections method for solving a consistent system of linear equations

$$\langle a_i, x \rangle = b_i, i = 1, \ldots, m. \tag{8}$$

His method, starting from an arbitrary point $x^{(0,m)} \in \mathbb{R}$, calculates the following sequence of point groups:

$$x^{(k,1)} = \pi_1\left(x^{(k-1,m)}\right), x^{(k,2)} = \pi_2\left(x^{(k,1)}\right), \ldots, x^{(k,m)} = \pi_m\left(x^{(k,m-1)}\right) \tag{9}$$

for $k = 1, 2, 3, \ldots$ Here, π_i $(i = 1, \ldots, m)$ is the orthogonal projection onto hyperplane H_i defined by equation (6). This sequence converges to the solution of system (8). Geometrically, the method can be interpreted as follows. The initial point $x^{(0,m)}$ is projected orthogonally onto hyperplane H_1. The projection is the point $x^{(1,1)}$, which now is thrown onto H_2. The resulting point $x^{(1,2)}$ is then thrown onto H_3 and gives the point $x^{(1,3)}$, etc. As a result, we obtain the last point $x^{(1,m)}$ from the first point group. The second point group is constructed in the same way, starting from the point $x^{(1,m)}$. The process is repeated for $k = 2, 3, \ldots$

Cimmino proposed in [57] (English description in [58]) a simultaneous projection method for the same problem. This method uses the following *orthogonal reflection* operation

$$\rho_i(x) = x - 2\frac{\langle a_i, x \rangle - b_i}{\|a_i\|^2} a_i, \tag{10}$$

which calculates the point $\rho_i(x)$ symmetric to the point x with respect to the hyperplane H_i. For the current approximation $x^{(k)}$, the Cimmino method simultaneously calculates reflections with respect to all hyperplanes H_i $(i = 1, \ldots, m)$, and then a convex combination of these reflections is used to form the next approximation:

$$x^{(k+1)} = \sum_{i=1}^{m} w_i \rho_i\left(x^{(k)}\right), \tag{11}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

where $w_i > 0$ $(i = 1, \ldots, m)$, $\sum_{i=1}^{m} w_i = 1$. When $w_i = \frac{1}{m}$ $(i = 1, \ldots, m)$, equation (11) is transformed into the following equation:

$$x^{(k+1)} = \frac{1}{m} \sum_{i=1}^{m} \rho_i \left(x^{(k)} \right). \tag{12}$$

Agmon [59] and Motzkin and Schoenberg [60] generalized the projection method from equations to inequalities. To solve problem (1), they introduce the *relaxed projection*

$$\pi_i^{\lambda}(x) = (1 - \lambda)x + \lambda \pi_i(x), \tag{13}$$

where $0 < \lambda < 2$. It is obvious that $\pi_i^1(x) = \pi_i(x)$. To calculate the next approximation, relaxed projection method uses the following equation:

$$x^{(k+1)} = \pi_l^{\lambda} \left(x^{(k)} \right), \tag{14}$$

where

$$\left\| x^{(k)} - \pi_l \left(x^{(k)} \right) \right\| = \max_{x^{(k)} \notin \hat{H}_i} \left\| x^{(k)} - \pi_i \left(x^{(k)} \right) \right\|. \tag{15}$$

Informally, the next approximation $x^{(k+1)}$ is a relaxed projection of the previous approximation $x^{(k)}$ with respect to the furthest hyperplane H_l bounding the half-space $\hat{H}_l$ not containing $x^{(k)}$. Agmon in [59] showed that sequence $x^{(k)}$ converges, as $k \to \infty$, to a point on the boundary of M.

Censor and Elfving, in [61], generalized the Cimmino method to the case of linear inequalities. They consider the relaxed projection onto the half-space $\hat{H}_i$ defined as follows:

$$\hat{\pi}_i^{\lambda}(x) = (1 - \lambda)x - \lambda \frac{\max\{0, \langle a_i, x \rangle - b_i\}}{\|a_i\|^2} a_i \tag{16}$$

that gives the equation

$$x^{(k+1)} = \sum_{i=1}^{m} w_i \hat{\pi}_i^{\lambda} \left(x^{(k)} \right). \tag{17}$$

Here, $0 < \lambda < 2$, and $w_i > 0$ $(i = 1, \ldots, m)$, $\sum_{i=1}^{m} w_i = 1$. In [62], De Pierro proposed an approach to convergence proof for this method, which differs from the approach of Censor and Elfving. De Piero's approach is also acceptable for the case when the underlying system of linear inequalities is infeasible. In this case, for $\lambda = 1$, sequence (17) converges to the point that is the minimum of the function $f(x) = \sum_{i=1}^{m} w_i \|\hat{\pi}_i(x) - x\|^2$, i.e., it is a weighted (with the weights w_i) least squares solution of system (1).

The Cimmino-like methods allow efficient parallelization, since orthogonal projections (reflections) can be calculated simultaneously and independently. The article [63] investigates the efficiency of parallelization of the Cimmino-like method on Xeon Phi manycore processors. In [64], the scalability of the Cimmino method for multiprocessor systems with distributed memory is evaluated. The applicability of the Cimmino-like method for solving non-stationary systems of linear inequalities on computing clusters is considered in [65].

As a recent work, we can mention article [66], which extends the Agmon-Motzkin-Schoenberg relaxation method for the case of semi-infinite inequality systems. The authors consider the system with an infinite number of inequalities in the finite-dimensional Euclidean space $\mathbb{R}^n$:

$$\langle a_i, x \rangle \leqslant b_i, i \in I, \tag{18}$$

where I is an arbitrary infinite index set. The main idea of the method is as follows. Let the hyperplane $H_x^{(\infty)} = \sup\{\max(\langle a_i, x \rangle - b_i, 0) | i \in I\}$ be the *biggest violation* with respect to x. Let $x^{(0)}$ be an arbitrary initial point. If the current iteration $x^{(k)}$ is not a solution

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

of system (18), then let $x^{(k+1)}$ be the orthogonal projection of $x^{(k)}$ onto a hyperplane H_i ($i \in I$) near the biggest violation $H^{(\infty)}_{x^{(k)}}$. If system (18) is consistent, then the sequence $\left\{x^{(k)} | k = 1, 2, \dots\right\}$ generated by the described method converges to the solution of this system.

Solving systems of linear inequalities is closely related to LP problems, so projection-type methods can be effectively used to solve this class of problems. The equivalence of the linear feasibility problem and the LP problem is based on the primal-dual LP problem. Consider the primal LP problem in the matrix form:

$$\bar{x} = \arg\max_x \{\langle c, x\rangle | Ax \leqslant b, x \geqslant \mathbf{0}\}, \tag{19}$$

where $c, x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m\times n}$, and $c \neq \mathbf{0}$. Here and below, $\langle \cdot, \cdot \rangle$ stands for the dot product of vectors. Let us construct the dual problem with respect to problem (19):

$$\bar{u} = \arg\min_u \left\{ \langle b, u\rangle | A^T u \geqslant c, u \geqslant \mathbf{0} \right\}, \tag{20}$$

where $u \in \mathbb{R}^m$. The following primal-dual equality holds:

$$\langle c, \bar{x}\rangle = \max_{Ax\leqslant b, x\geqslant \mathbf{0}} \langle c, x\rangle = \min_{A^T u\geqslant c, u\geqslant \mathbf{0}} \langle b, u\rangle = \langle b, \bar{u}\rangle. \tag{21}$$

In [67,68], Eremin proposed the following method based on the primal-dual approach. Let the inequality system

$$A'x \leqslant b' \tag{22}$$

define the feasible region of primal problem (19). This system is obtained by adding to the system $Ax \leqslant b$ the vector inequality $-x \leqslant 0$. In this case, $A' \in \mathbb{R}^{(m+n)\times n}$, and $b' \in \mathbb{R}^{m+n}$. Let a'_i stand the ith row of the matrix A'. For each inequality $\langle a'_i, x\rangle \leqslant b'_i$, define the closed half space

$$\hat{H}'_i = \left\{x \in \mathbb{R}^n \middle| \langle a'_i, x\rangle \leqslant b'_i\right\}, \tag{23}$$

and its bounding hyperplane

$$H'_i = \left\{x \in \mathbb{R}^n \middle| \langle a'_i, x\rangle = b'_i\right\}. \tag{24}$$

Let $\pi'_i(x)$ stand the orthogonal projection of point x onto the hyperplane H'_i:

$$\pi'_i(x) = x - \frac{\langle a'_i, x\rangle - b'_i}{\left\|a'_i\right\|^2} a'_i. \tag{25}$$

Let us define the projection onto the half-space $\hat{H}'_i$:

$$\hat{\pi}'_i(x) = x - \frac{\max\{0, \langle a'_i, x\rangle - b'_i\}}{\left\|a'_i\right\|^2} a'_i. \tag{26}$$

This projection has the following two properties:

$$x \notin \hat{H}'_i \Rightarrow \hat{\pi}'_i(x) = \pi'_i(x); \tag{27}$$

$$x \in \hat{H}'_i \Rightarrow \hat{\pi}'_i(x) = x. \tag{28}$$

Define $\varphi_1 : \mathbb{R}^n \to \mathbb{R}^n$ as follows:

$$\varphi_1(x) = \frac{1}{m+n} \sum_{i=1}^{m+n} \hat{\pi}'_i(x). \tag{29}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

In the same way, define the feasible region of dual problem (20) as follows:

$$D'x \geqslant c', \tag{30}$$

where $D = A^T \in \mathbb{R}^{n\times m}$, $D' \in \mathbb{R}^{(m+n)\times m}$, and $c' \in \mathbb{R}^{n+m}$. Denote

$$\hat{\eta}'_j(u) = u - \frac{\max\left\{0, \left\langle d'_j, u \right\rangle - c'_j\right\}}{\left\|d'_j\right\|^2} d'_j, \tag{31}$$

and define $\varphi_2 : \mathbb{R}^m \to \mathbb{R}^m$ as follows:

$$\varphi_2(u) = \frac{1}{n+m} \sum_{j=1}^{n+m} \hat{\eta}'_j(x). \tag{32}$$

Now, define $\varphi_3 : \mathbb{R}^{n+m} \to \mathbb{R}^{n+m}$ as follows:

$$\varphi_3([x,u]) = [x,u] - \frac{\langle c,x\rangle - \langle b,u\rangle}{\|c\|^2 + \|b\|^2}[c,-b], \tag{33}$$

which is corresponding to equation (21). Here, $[\cdot\,,\cdot]$ stands for the concatenation of vectors.

Finally, define $\varphi : \mathbb{R}^{n+m} \to \mathbb{R}^{n+m}$ as follows:

$$\varphi([x,u]) = \varphi_3(\varphi_1(x), \varphi_2(u)). \tag{34}$$

If the feasible region of the primal problem is a bounded and nonempty set, then the sequence

$$\left[x^{(k+1)}, u^{(k+1)}\right] = \varphi\left(\left[x^{(k)}, u^{(k)}\right]\right) \tag{35}$$

converges to the point $[\bar{x}, \bar{u}]$, where $\bar{x}$ is the solution of primal problem (19), and $\bar{u}$ is the solution of dual problem (20).

Article [69] proposes a method for solving non-degenerate LP problems based on calculating the orthogonal projection of some special point, independent of the main part of data describing the LP problem, onto a problem-dependent cone generated by the constraint inequalities. Actually, this method solves a symmetric positive definite system of linear equations of a special kind. The author demonstrates a finite algorithm of active-set family that is capable of calculating orthogonal projections for problems with up to thousands of rows and columns. The main drawback of this method is a significant increasing the dimension of the primary problem.

In article [70], Censor proposes the linear superiorization (LinSup) method as a tool for solving LP problems. The LinSup method does not guarantee finding the minimum point of the LP problem, but it directs the linear feasibility-seeking algorithm that it uses toward a point with a decreasing value of the objective function. This process is not identical with that employed by LP solvers but it is a possible alternative to the Simplex method for problems of huge size. The basic idea of LinSup is to add an extra term, called *perturbation term*, to the iterative equation of the projection method. The perturbation term steers the feasibility-seeking algorithm toward reduced the objective function values. In the case of LP problem (19), the objective function is $f(x) = \langle c, x\rangle$, and LinSup adds $\left(-\eta \frac{c}{\|c\|}\right)$ as a perturbation term to iterative equation (17):

$$x^{(k+1)} = \left(-\eta \frac{c}{\|c\|}\right) + \sum_{i=1}^{m} w_i \hat{\pi}_i^{\lambda}\left(x^{(k)}\right). \tag{36}$$

Here, $0 < \eta < 1$ is a *perturbation parameter*.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

Article [52] proposes a mathematical model for the visual representation of multidimensional LP problems. To visualize a feasible LP problem, an objective hyperplane H_c is introduced, the normal to which is the gradient of the objective function $f(x) = \langle c, x \rangle$. In the case of seeking the maximum, the objective hyperplane is positioned in such a way that the value of the objective function at all its points is greater then the value of the objective function at all points of the convex polytope M, which is the feasible region of the LP problem. For any point $g \in H_c$, the *objective projection* $\gamma_M(g)$ onto M is defined as follows:

$$\gamma_M(g) = \begin{cases} \arg\min\limits_x \{ \|x - g\| | x \in M, \pi_{H_c}(x) = g \}, \text{if } \exists x \in M : \pi_{H_c}(x) = g; \\ +\infty, \text{if } \neg \exists x \in M : \pi_{H_c}(x) = g. \end{cases} \tag{37}$$

Here, $\pi_{H_c}(x)$ denotes the orthogonal projection onto H_c. On the objective hyperplane H_c, a rectangular lattice of points $\mathfrak{G} \in \mathbb{R}^n \times \mathbb{R}^{K^{(n-1)}}$ is constructed, where K is the number of lattice points in one dimension. Each point $g \in \mathfrak{G}$ is mapped to the real number $\|\gamma_M(g) - g\|$. This mapping generates a matrix of dimension $(n-1)$, which is an *image of the LP problem*. This approach opens up the possibility of using feed-forward artificial neural networks, including convolutional neural networks, to solve multidimensional LP problems. One of the main obstacles to the implementation of this approach is the problem of generating a training set. The literature review shows that there is no suitable method capable of constructing such a training set compatible with the described approach. In the next sections, we present such a method.

3. Theoretical Basis

In this section we present a theoretical basis used to construct the apex method. Consider the LP problem in the following form:

$$\bar{x} = \arg\max_{x \in \mathbb{R}^n} \{ \langle c, x \rangle | Ax \leqslant b \}, \tag{38}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$, $m > 1$, and $c \neq \mathbf{0}$. 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:

$$\begin{gathered} -x_1 \leqslant 0; \\ \cdots \\ -x_n \leqslant 0. \end{gathered}$$

Let $\mathcal{P}$ stand for the set of row indices in matrix A:

$$\mathcal{P} = \{1, \cdots, m\}. \tag{39}$$

Let $a_i \in \mathbb{R}^n$ be a vector formed by the elements of the ith row of the matrix A, and $a_i \neq \mathbf{0}$ for all $i \in \mathcal{P}$. We denote by $\hat{H}_i$ the closed half space defined by the inequality $\langle a_i, x \rangle \leqslant b_i$, and by H_i the hyperplane bounding $\hat{H}_i$:

$$\hat{H}_i = \{ x \in \mathbb{R}^n | \langle a_i, x \rangle \leqslant b_i \}; \tag{40}$$

$$H_i = \{ x \in \mathbb{R}^n | \langle a_i, x \rangle = b_i \}. \tag{41}$$

Definition 1. *The half-space $\hat{H}$ is called dominant with respect to the vector c, or briefly c-dominant, if*

$$\forall x \in \hat{H}, \forall \lambda \in \mathbb{R}_{>0} : x + \lambda c \in \hat{H}. \tag{42}$$

The geometric meaning of this definition is that a ray outgoing from a point belonging to a half-space in the direction of vector c belongs to this half-space.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

Definition 2. *The half-space $\hat{H}$ is called recessive with respect to the vector c, or briefly c-recessive, if it is not c-dominant, i.e.,*

$$\forall x \in \hat{H}, \exists \lambda \in \mathbb{R}_{>0} : x + \lambda c \notin \hat{H}. \tag{43}$$

The following proposition provides the necessary and sufficient condition for the c-recessivity of the half-space.

Proposition 1. *Let a half-space $\hat{H}$ be defined by the following equation:*

$$\hat{H} = \{x \in \mathbb{R}^n | \langle a, x \rangle \leqslant \beta\}. \tag{44}$$

Then, the necessary and sufficient condition for the c-recessivity of the half-space $\hat{H}$ is

$$\langle a, c \rangle > 0. \tag{45}$$

Proof. Let us prove the necessity first. Let condition (43) hold. Denote

$$x' = \frac{\beta a}{\|a\|^2}. \tag{46}$$

It follows

$$\langle a, x' \rangle = \left\langle a, \frac{\beta a}{\|a\|^2} \right\rangle = \beta \frac{\langle a, a \rangle}{\|a\|^2} = \beta, \tag{47}$$

i.e., $x' \in \hat{H}$. By virtue of (43), there is $\lambda' \in \mathbb{R}_{>0}$ such that

$$x' + \lambda' c \notin \hat{H}, \tag{48}$$

i.e.,

$$\langle a, x' + \lambda' c \rangle > \beta. \tag{49}$$

Substituting the right-hand side of equation (46) instead of x', we obtain

$$\left\langle a, \frac{\beta a}{\|a\|^2} + \lambda' c \right\rangle > \beta. \tag{50}$$

Since $\lambda' > 0$, it follows

$$\langle a, c \rangle > 0. \tag{51}$$

Thus, the necessity is proved.

Let us prove the sufficiency by contradiction. Assume that (45) holds, and $\hat{H}$ is not c-recessive, i.e.,

$$\forall x \in \hat{H}, \forall \lambda \in \mathbb{R}_{>0} : x + \lambda c \in \hat{H}. \tag{52}$$

Since x' defined by (46) belongs to $\hat{H}$, it follows

$$x' + \lambda c \in \hat{H} \tag{53}$$

for all $\lambda \in \mathbb{R}_{>0}$, i.e.,

$$\langle a, x' + \lambda c \rangle \leqslant \beta. \tag{54}$$

Substituting the right-hand side of equation (46) instead of x', we obtain

$$\left\langle a, \frac{\beta a}{\|a\|^2} + \lambda c \right\rangle \leqslant \beta. \tag{55}$$

Since $\lambda > 0$, it follows

$$\langle a, c \rangle \leqslant 0. \tag{56}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

But this contradicts (45). □

Denote

$$e_c = \frac{c}{\|c\|}, \tag{57}$$

i.e., e_c stands for the unit vector parallel to vector c.

Proposition 2. *Let the half-space $\hat{H}_i$ be c-recessive. Then, for any point $x' \in \mathbb{R}$ and any number $\eta > 0$, the point*

$$z = x' + \left(\eta + \frac{b_i - \langle a_i, x' \rangle}{\langle a_i, e_c \rangle}\right) e_c \tag{58}$$

does not belong to the half-space $\hat{H}_i$, i.e.

$$\langle a_i, z \rangle > b_i. \tag{59}$$

Proof. The half-space $\hat{H}_i$ is c-recessive, therefore, according to Proposition 1, the following inequality holds:

$$\langle a_i, c \rangle > 0. \tag{60}$$

Taking (58) into account, we have

$$\langle a_i, z \rangle = \left\langle a_i, x' + \left(\eta + \frac{b_i - \langle a_i, x' \rangle}{\langle a_i, e_c \rangle}\right) e_c \right\rangle = \eta \langle a_i, e_c \rangle + b_i. \tag{61}$$

Substituting the right-hand side of equation (57) instead of e_c in (61), we obtain

$$\langle a_i, z \rangle = \frac{\eta}{\|c\|} \langle a_i, c \rangle + b_i. \tag{62}$$

Since $\eta > 1$, by virtue of (60), the inequality $\frac{\eta}{\|c\|} \langle a_i, c \rangle > 0$ holds. It follows that $\langle a_i, z \rangle > b_i$, i.e. $z \notin \hat{H}_i$. □

Define

$$\mathcal{I}_c = \{ i \in \mathcal{P} | \langle a_i, c \rangle > 0 \}, \tag{63}$$

i.e., $\mathcal{I}_c$ is the set of indices for which the half-space $\hat{H}_i$ is c-recessive. We assume from now on that

$$\mathcal{I}_c \neq \varnothing. \tag{64}$$

Corollary 1. *Let an arbitrary feasible point x' of LP problem (38) be given:*

$$\forall i \in \mathcal{P} : \langle a_i, x' \rangle \leqslant b_i. \tag{65}$$

Then, for any positive number $\eta \in \mathbb{R}_{>0}$, the point

$$z = x' + \left(\eta + \max\left\{ \frac{b_i - \langle a_i, x' \rangle}{\langle a_i, e_c \rangle} \middle| i \in \mathcal{I}_c \right\}\right) e_c \tag{66}$$

does not belong to any c-recessive half-space $\hat{H}_i$, i.e.,

$$\forall i \in \mathcal{I}_c : \langle a_i, z \rangle > b_i. \tag{67}$$

Proof. From (65), we obtain

$$\forall i \in \mathcal{I}_c : b_i - \langle a_i, x' \rangle \geqslant 0. \tag{68}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

According to (63) and (57), the following condition holds:

$$\forall i \in \mathcal{I}_c : \langle a_i, e_c \rangle > 0. \tag{69}$$

Hence,

$$\max\left\{ \frac{b_i - \langle a_i, x' \rangle}{\langle a_i, e_c \rangle} \middle| i \in \mathcal{I}_c \right\} \geqslant 0 \tag{70}$$

for any $i \in \mathcal{I}_c$. Fix any $j \in \mathcal{I}_c$, and define

$$\eta' = \eta + \max\left\{ \frac{b_i - \langle a_i, x' \rangle}{\langle a_i, e_c \rangle} \middle| i \in \mathcal{I}_c \right\} - \frac{b_j - \langle a_j, x' \rangle}{\langle a_j, e_c \rangle}, \tag{71}$$

where $\eta > 0$. Taking into account (70), it follows that $\eta' > 0$. Using (66) and (71), we obtain

$$z = x' + \left(\eta + \max\left\{ \frac{b_i - \langle a_i, x' \rangle}{\langle a_i, e_c \rangle} \middle| i \in \mathcal{I}_c \right\} \right) e_c = x' + \left(\eta' + \frac{b_j - \langle a_j, x' \rangle}{\langle a_j, e_c \rangle} \right) e_c. \tag{72}$$

According to Proposition 2, it follows that $\langle a_j, z \rangle > b_j$, i.e., the point z defined by (66) does not belong to the half-space $\hat{H}_j$ for any $j \in \mathcal{I}_c$. □

The following proposition specifies the region containing a solution of LP problem (38).

Proposition 3. *Let $\bar{x}$ be a solution of LP problem (38). Then, there is an index $i' \in \mathcal{I}_c$ such that*

$$\bar{x} \in H_{i'}, \tag{73}$$

i.e., there is a c-recessive half-space $\hat{H}_{i'}$ such that its bounding hyperplane $H_{i'}$ includes $\bar{x}$.

Proof. Denote by $\mathcal{J}_c$ the set of indices for which the half-space $\hat{H}_j$ is c-dominant:

$$\mathcal{J}_c = \mathcal{P} \backslash \mathcal{I}_c. \tag{74}$$

Since $\bar{x}$ belongs to the feasible region of LP problem (38), then

$$\bar{x} \in \bigcap_{j \in \mathcal{J}_c} \hat{H}_j, \tag{75}$$

and

$$\bar{x} \in \bigcap_{i \in \mathcal{I}_c} \hat{H}_i. \tag{76}$$

Define the ray Y as follows:

$$Y = \{ \bar{x} + \lambda c | \lambda \in \mathbb{R}_{\geqslant 0} \}. \tag{77}$$

By Definition 1, we have

$$Y \subset \bigcap_{j \in \mathcal{J}_c} \hat{H}_j, \tag{78}$$

i.e., the ray Y belongs to the all c-dominant half-spaces. By virtue of Definition 2,

$$\forall i \in \mathcal{I}_c, \exists \lambda \in \mathbb{R}_{>0} : \bar{x} + \lambda c \notin \hat{H}_i. \tag{79}$$

Taking into account (76), it means that

$$\forall i \in \mathcal{I}_c : Y \cap H_i = y_i \in \mathbb{R}^n, \tag{80}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

i.e., the intersection of the ray Y and any hyperplane H_i bounding the c-recessive half-space $\hat{H}_i$ is a single point $y_i \in \mathbb{R}^n$. Let

$$i' = \arg\min_{i \in \mathcal{I}_c} \{ \|\bar{x} - y_i\| | y_i = Y \cap H_i \}, \tag{81}$$

i.e., $H_{i'}$ is the nearest hyperplane to the point $\bar{x}$ for $i' \in \mathcal{I}_c$. Denote by $\bar{y}$ the intersection of the ray Y and the hyperplane $H_{i'}$:

$$\bar{y} = Y \cap H_{i'}. \tag{82}$$

According to (81),

$$\bar{y} \in \bigcap_{i \in \mathcal{I}_c} \hat{H}_i, \tag{83}$$

i.e., the point $\bar{y}$ belongs to the all c-recessive half-spaces. By (78), it follows that

$$\bar{y} \in \bigcap_{i \in \mathcal{P}} \hat{H}_i, \tag{84}$$

i.e., $\bar{y}$ belongs to the feasible region of LP problem (38). Let

$$\lambda' = \|\bar{x} - \bar{y}\|. \tag{85}$$

Then, in virtue of (77),

$$\langle c, \bar{y} \rangle = \langle c, \bar{x} + \lambda' e_c \rangle = \langle c, \bar{x} \rangle + \lambda' \frac{\langle c, c \rangle}{\|c\|} = \langle c, \bar{x} \rangle + \lambda' \|c\|. \tag{86}$$

Since $\bar{x}$ is a solution of LP problem (38), the following condition holds:

$$\forall y \in \bigcap_{i \in \mathcal{P}} \hat{H}_i : \langle c, y \rangle \leqslant \langle c, \bar{x} \rangle. \tag{87}$$

Comparing this with (84), we obtain that

$$\langle c, \bar{y} \rangle \leqslant \langle c, \bar{x} \rangle. \tag{88}$$

Taking into account that $\lambda' \geqslant 0$ and $c \neq \mathbf{0}$, by virtue (86) and (88), we obtain $\lambda' = 0$. By (85), it follows that $\bar{x} = \bar{y}$. By (82), this means that $\bar{x} \in H_{i'}$, where $\hat{H}_{i'}$ is a c-recessive half-space. □

Definition 3. *Let $M \neq \emptyset$ be a convex closed set. A single-valued mapping $\varphi : \mathbb{R}^n \to \mathbb{R}^n$ is called M-Fejér mapping [67], if*

$$\forall x \in M : \varphi(x) = x, \tag{89}$$

and

$$\forall x \notin M, \forall y \in \mathbb{R}^n : \|\varphi(x) - y\| < \|x - y\|. \tag{90}$$

Proposition 4. *Let $x^{(0)} \in \mathbb{R}^n$. If $\varphi(\cdot)$ is a continuous M-Fejér mapping and*

$$\left\{ x^{(k)} = \varphi^k \left(x^{(0)} \right) \right\}_{k=1}^{\infty}$$

is the iterative process generated by this mapping, then

$$x^{(k)} \to \tilde{x} \in M. \tag{91}$$

Proof. The convergence follows directly from Theorem 6.2 and Corollary 6.3 in [67]. □

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

Let $\pi_i(x)$ stand for the orthogonal projection of point x onto hyperplane H_i:

$$\pi_i(x) = x - \frac{\langle a_i, x\rangle - b_i}{\|a_i\|^2} a_i. \tag{92}$$

The next proposition provides a continuous M-Fejér mapping, which will be used in the apex method.

Proposition 5. *Let $M \neq \emptyset$ be the convex closed set representing the feasible region of LP problem (38):*

$$M = \bigcap_{i=1}^{m} \hat{H}_i. \tag{93}$$

For any point $x \in \mathbb{R}^n$, let us define

$$\mathcal{J}_x = \{i | \langle a_i, x\rangle > b_i; i \in \mathcal{P}\}, \tag{94}$$

i.e., $\mathcal{J}_x$ is the set of indices for which the half-space $\hat{H}_i$ does not contain the point x. Then, the single-valued mapping $\psi : \mathbb{R}^n \to \mathbb{R}^n$ defined by the equation

$$\psi(x) = \begin{cases} x, & \text{if } x \in M; \\ \frac{1}{|\mathcal{J}_x|} \sum\limits_{i \in \mathcal{J}_x} \pi_i(x), & \text{if } x \notin M. \end{cases} \tag{95}$$

is a continuous M-Fejér mapping.

Proof. Obviously, the mapping $\psi(\cdot)$ is continuous. Let us prove that condition (90) holds. Our proof is based on a general scheme presented in [67]. Let $y \in M$, and $x \notin M$. It follows that

$$\mathcal{J}_x \neq \emptyset. \tag{96}$$

By virtue of the equation (94), the following inequalities holds

$$\|\pi_i(x) - x\| > 0 \tag{97}$$

for all $i \in \mathcal{J}_x$. According to Lemma 3.13 in [67], the following inequality holds for all $i \in \mathcal{J}_x$:

$$\|\pi_i(x) - y\|^2 \leqslant \|x - y\|^2 - \|\pi_i(x) - x\|^2. \tag{98}$$

It follows that

$$\|y - \psi(x)\|^2 = \left\| y - \tfrac{1}{|\mathcal{J}_x|} \sum_{i \in \mathcal{J}_x} \pi_i(x) \right\|^2 = \left\| \tfrac{1}{|\mathcal{J}_x|} \sum_{i \in \mathcal{J}_x} (y - \pi_i(x)) \right\|^2 \leqslant \tfrac{1}{|\mathcal{J}_x|^2} \sum_{i \in \mathcal{J}_x} \|y - \pi_i(x)\|^2 \leqslant$$
$$\leqslant \tfrac{1}{|\mathcal{J}_x|} \sum_{i \in \mathcal{J}_x} \|y - \pi_i(x)\|^2 \leqslant \tfrac{1}{|\mathcal{J}_x|} \sum_{i \in \mathcal{J}_x} \left(\|x - y\|^2 - \|\pi_i(x) - x\|^2 \right) \leqslant$$
$$\leqslant \|x - y\|^2 - \tfrac{1}{|\mathcal{J}_x|} \sum_{i \in \mathcal{J}_x} \|\pi_i(x) - x\|^2.$$

According to (96) and (97) , the following inequality holds:

$$\tfrac{1}{|\mathcal{J}_x|} \sum_{i \in \mathcal{J}_x} \|\pi_i(x) - x\|^2 > 0. \tag{99}$$

Hence,

$$\forall x \notin M, \forall y \in \mathbb{R}^n : \|\psi(x) - y\| < \|x - y\|.$$

□

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

Definition 4. *Let $M \neq \emptyset$ be the feasible region of LP problem (38), $\psi(\cdot)$ be the mapping defined by equation (95). The pseudoprojection $\rho_M(x)$ of the point x onto the feasible polytope M is the limit point of the sequence $[x, \psi(x), \psi^2(x), \ldots, \psi^k(x), \ldots]$:*

$$\lim_{k\to\infty} \left\| \rho_M(x) - \psi^k(x) \right\| = 0. \tag{100}$$

The correctness of this definition is ensured by propositions 4 and 5.

4. Description of Apex Method

In this section, we describe a new scalable iterative method for solving LP problem (38), called the "apex method". The apex method is based on the predictor-corrector framework and proceeds in two stages: *quest* (predictor) and *target* (corrector). The quest stage calculates a rough initial approximation of LP problem (38). The target stage refines the initial approximation with a given precision. The main operation used in the apex method is an operation that calculates a pseudoprojection according to Definition 4. This operation is used both in the quest stage and in the target stage. In the next section, we describe and investigate a parallel algorithm for calculating a pseudoprojection.

4.1. Algorithm for Calculating Pseudoprojection

In this section, we consider the implementation of the pseudoprojection operation in the form of sequential and parallel algorithms. The pseudoprojection operation $\rho_M(\cdot)$ maps an arbitrary point $x \in \mathbb{R}^n$ to a point $\rho_M(x)$ belonging to the feasible polytope M, which is the feasible region of LP problem (38). The calculation of $\rho_M(x)$ is organized as an iterative process using equation (95). A sequential implementation of this process is presented by Algorithm 1.

Algorithm 1 Calculating the pseudoprojection $\rho_M(x)$

Require: $\hat{H}_i = \{x \in \mathbb{R}^n | \langle a_i, x\rangle \leqslant b_i\}$, $M = \bigcap_{i=1}^m \hat{H}_i$, $M \neq \emptyset$

function $\rho_M(x)$
  $k := 0$
  $x^{(0)} := x$
  repeat
    $\mathcal{J} := \emptyset$
    for $i = 1 \ldots m$ **do**
      if $\left\langle a_i, x^{(k)} \right\rangle > b_i$ **then**
        $\mathcal{J} := \mathcal{J} \cup \{i\}$
      end if
    end for
    if $\mathcal{J} = \emptyset$ **then**
      return $x^{(k)}$
    end if
    $S := 0$
    for all $i \in \mathcal{J}$ **do**
      $S := S + \left(\left\langle a_i, x^{(k)} \right\rangle - b_i\right) a_i / \|a_i\|^2$
    end for
    $x^{(k+1)} := x^{(k)} - S/|\mathcal{J}|$
    $k := k + 1$
  until $\left\| x^{(k)} - x^{(k-1)} \right\| < \epsilon$
  return $x^{(k)}$
end function

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

Let us give brief comments on this implementation. The main iterative process of constructing a sequence of Fejér's approximations is represented by the **repeat/until** loop implemented in steps 4–20. The steps 5–10 calculate the set $\mathcal{J}$ of indices of half-spaces $\hat{H}_i$ violated by the point $x^{(k)}$ presenting the current approximation. In steps 14–18, the next approximation $x^{(k+1)}$ is calculated by equation (95). The process terminates when the distance between adjacent approximations becomes less than ϵ, where ϵ is a small positive parameter. Computational experiments (see [1,71]) show that, in the case of large LP problems, the calculation of a pseudoprojection is a process with high computational complexity. Therefore, we developed a parallel implementation of Algorithm 1, presented by Algorithm 2.

Algorithm 2 Parallel calculation of a pseudoprojection

	Master		***l*th Worker (*l*=0,...,*L*-1)**
1:	**input** $n, x^{(0)}$	1:	**input** n, m, A, b, c
2:		2:	$L :=$ NumberOfWorkers
3:	$k := 0$	3:	$\mathcal{L}_{map(l)} := [lm/L, \ldots, ((l+1)m/L) - 1]$
4:	**repeat**	4:	**repeat**
5:	**Bcast** $x^{(k)}$	5:	**RecvFromMaster** $x^{(k)}$
6:		6:	$\mathcal{L}_{reduce(l)} := Map(\mathrm{F}_{x^{(k)}}, \mathcal{L}_{map(l)})$
7:		7:	$(u_l, \sigma_l) := Reduce(\oplus, \mathcal{L}_{reduce(l)})$
8:	**Gather** $\mathcal{L}_{reduce}$	8:	**SendToMaster** (u_l, σ_l)
9:	$(u, \sigma) := Reduce(\oplus, \mathcal{L}_{reduce})$	9:	
10:	$x^{(k+1)} := u/\sigma$	10:	
11:	$k := k + 1$	11:	
12:	$exit := \left\|x^{(k)} - x^{(k-1)}\right\| < \epsilon$	12:	
13:	**Bcast** $exit$	13:	**RecvFromMaster** $exit$
14:	**until** $exit$	14:	**until** $exit$
15:	**output** $x^{(k)}$	15:	
16:	**stop**	16:	**stop**

Algorithm 2 is based on the BSF parallel computation model [72] designed for a cluster computing system. The BSF model 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*. In Algorithm 2, we use the list $\mathcal{L}_{map} = [1, \ldots, m]$ of ordinal numbers of constraints of LP problem (38) as the second parameter of the higher-order function *Map*. As the first parameter of the higher-order function *Map*, we use the parameterized function

$$\mathrm{F}_x : \mathcal{P} \to \mathbb{R}^n \times \mathbb{Z}_{\geqslant 0}$$

defined as follows:

$$\begin{aligned} &\mathrm{F}_x(i) = (u_i, \sigma_i); \\ &u_i = \begin{cases} \pi_i(x), & \textbf{if } \langle a_i, x\rangle > b_i; \\ \mathbf{0}, & \textbf{if } \langle a_i, x\rangle \leqslant b_i; \end{cases} \\ &\sigma_i = \begin{cases} 1, & \textbf{if } \langle a_i, x\rangle > b_i; \\ 0, & \textbf{if } \langle a_i, x\rangle \leqslant b_i. \end{cases} \end{aligned} \tag{101}$$

Thus, the higher-order function $Map(\mathrm{F}_x, \mathcal{L}_{map})$ transforms the list $\mathcal{L}_{map}$ of constraint numbers into a list of pairs (u_i, σ_i). Here, u_i is the orthogonal projection of the point x onto

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

the hyperplane H_i in the case $x \notin \hat{H}_i$, and the zero vector otherwise; σ_i is the indicator that x violates the half-space $\hat{H}_i$ ($i = 1, \ldots, m$):

$$Map\big(\mathrm{F}_x, \mathcal{L}_{map}\big) = [\mathrm{F}_x(1), \ldots, \mathrm{F}_x(m)] = [(u_1, \sigma_1), \ldots, (u_m, \sigma_m)]. \tag{102}$$

Denote $\mathcal{L}_{reduce} = [(u_1, \sigma_1), \ldots, (u_m, \sigma_m)]$. Define a binary associative operation

$$\oplus : \mathbb{R}^n \times \mathbb{Z}_{\geqslant 0} \to \mathbb{R}^n \times \mathbb{Z}_{\geqslant 0},$$

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

$$(u', \sigma') \oplus (u'', \sigma'') = (u' + u'', \sigma' + \sigma''). \tag{103}$$

The higher-order function $Reduce(\oplus, \mathcal{L}_{reduce})$ folds the list $\mathcal{L}_{reduce}$ into the single pair by sequentially applying the operation $\oplus$ to all elements of the list:

$$Reduce(\oplus, \mathcal{L}_{reduce}) = (u_1, \sigma_1) \oplus \ldots \oplus (u_m, \sigma_m) = (u, \sigma), \tag{104}$$

where

$$u = \sum_{i=1}^{m} u_i; \tag{105}$$

$$\sigma = \sum_{i=1}^{m} \sigma_i. \tag{106}$$

In Algorithm 2, a parallel execution of work is organized according to the master/worker scheme. The parallel algorithm includes $L + 1$ processes: one master process and L worker processes. The master manages the computations, distributes the work among the workers, gathers the results back from them and summarizes all the results to obtain the final result. For the sake of simplicity, it is assumed that the number of constraints m of LP problem (38) is a multiple of the number of workers L. In Step 1, the master reads the space dimension n and the staring point $x^{(0)}$. Step 3 of the master assigns zero to the iteration counter k. Steps 4–14 implement the main **repeat/until** loop calculating the pseudoprojection. In Step 5, master broadcasts the current approximation $x^{(k)}$ to all workers. Step 9 of the master gathers partial results from all workers. In Step 9, the master folds the partial results into the pair (u, σ), which is used to calculate the next approximation $x^{(k+1)}$ in Step 10. Step 11 of the master increases the iteration counter k by 1. In Step 12, the master calculates the criterion for stopping the iterative process and assigns the result to the Boolean variable *exit*. In Step 13, master broadcasts the value of the Boolean variable *exit* to all workers. In Step 14, the **repeat/until** loop ends if the Boolean variable *exit* takes the value *true*. Step 15 of the master outputs the last approximation $x^{(k)}$ as a result of the pseudoprojection. Step 16 terminates the master process.

All workers execute the same program codes, but with different data. In Step 1, the lth worker reads problem data. In steps 2–3, the lth worker defines its own sublist $\mathcal{L}_{map(l)}$ for processing. For convenience, we number the constraints starting from zero. The sublists of different workers do not overlap, and their concatenation represents the entire list to be processed:

$$\mathcal{L}_{map} = \mathcal{L}_{map(0)} +\!\!+ \ldots +\!\!+ \mathcal{L}_{map(L-1)}. \tag{107}$$

The **repeat/until** loop of the lth worker corresponds to the **repeat/until** loop of the master (steps 4–14). In Step 5, the lth worker receives the current approximation $x^{(k)}$ from the master. Step 6 of the lth worker executes the higher-order function *Map*, which applies the parameterized function $\mathrm{F}_{x^{(k)}}$ defined by (101) to all elements of the sublist $\mathcal{L}_{map(l)}$, resulting in the sublist $\mathcal{L}_{reduce(l)}$. Step 7 of the lth worker executes the higher-order function *Reduce*, which applies the operation $\oplus$ defined by (103) to all elements of the list $\mathcal{L}_{reduce(l)}$, resulting in the pair (u_l, σ_l). In Step 8, the lth worker sends its resulting pair (u_l, σ_l) to the master.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

In Step 13, the lth worker receives a value of the Boolean variable $exit$ from the master. If $exit = true$ then worker process is terminated. Otherwise, the **repeat/until** loop continues its work. The exchange operators **Bcast**, **Gather**, **RecvFromMaster**, and **SendToMaster** provide synchronization of the master and workers processes.

Let us estimate the scalability boundary of the described pseudoprojection parallel algorithm, using the cost metrics of BSF model [72]. Here, the *scalability boundary* refers to the number of worker processes at which the maximum speedup is achieved. The cost metric of the BSF model includes the following parameters.

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 current approximation $x^{(k)}$ to a single worker and receive the pair (u_l, σ_l) 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 $\oplus$.

According to equation (14) from [72], the scalability boundary L_{max} of a parallel algorithm can be estimated as follows:

$$L_{max} = \frac{1}{2}\sqrt{\left(\frac{t_c}{t_a \ln 2}\right)^2 + \frac{t_{Map}}{t_a} + 4m} - \frac{t_c}{t_a \ln 2}. \tag{108}$$

Let us calculate the time parameters in equation (108). To do this, we introduce the following notation for one iteration of the **repeat/until** loop implemented in steps 4–14 of Algorithm 2:

c_c : quantity of numbers sent from the master to the lth worker and back within one iteration;
c_F : quantity of arithmetic and comparison operations required to compute the function F_x defined by equations (101);
$c_\oplus$: quantity of arithmetic and comparison operations required to compute the binary operation $\oplus$.

In Step 5, the master sends to the lth worker one vector of dimension n. Then, in Step 8, the master receives a pair consisting of a vector of dimension n and a single number from the lth worker. In addition, in Step 13, the master sends a single Boolean value to the lth worker. Hence,

$$c_c = 2n + 1. \tag{109}$$

Taking into account equations (92), (101), and assuming that $\|a_i\|^2$ is calculated in advance, we obtain

$$c_F = 3n + 2. \tag{110}$$

According to (103), the following equations holds for $c_\oplus$:

$$c_\oplus = 2n + 1. \tag{111}$$

Let us denote by τ_{op} the execution time of one arithmetic or comparison operation, and by τ_{tr} the time of sending a single real number from one process to another (excluding latency). Then, using (109), (110) and (111), we obtain

$$t_c = c_c\tau_{tr} + 3D = (2n+1)\tau_{tr} + 3D; \tag{112}$$
$$t_{Map} = c_F m\tau_{op} = (3n+2)m\tau_{op}; \tag{113}$$
$$t_a = c_\oplus\tau_{op} = (2n+1)\tau_{op}. \tag{114}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

Recall that the parameter D denotes the latency. Substituting the right-hand sides of these equations into (108), we have

$$L_{max} = \frac{1}{2}\sqrt{\left(\frac{(2n+1)\tau_{tr}+3D}{(2n+1)\tau_{op}\ln 2}\right)^2 + \left(\frac{n+1}{2n+1}+5\right)m} - \frac{(2n+1)\tau_{tr}+3D}{(2n+1)\tau_{op}\ln 2},$$

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

$$L_{max} \approx O(\sqrt{m}). \tag{115}$$

This estimation suggests that Algorithm 2 is limited-scalable, and the scalability depends on the number of constraints m.

4.2. Quest Stage

The quest stage of the apex method plays the role of a predictor and includes the following steps.

1. Calculate a feasible point $\tilde{x} \in M$.
2. Calculate the apex point z.
3. Calculate the point $u^{(0)}$ that is the pseudoprojection of the apex point z onto the feasible polytope M.

The feasible point $\tilde{x}$, in Step 1, can be calculated by the following equation:

$$\tilde{x} = \begin{cases} \mathbf{0}, & \textbf{if } \mathbf{0} \in M; \\ \rho_M(\mathbf{0}), & \textbf{if } \mathbf{0} \notin M, \end{cases} \tag{116}$$

where $\rho_M(\cdot)$ is the operation of pseudoprojection onto the feasible polytope M (see Definition 4).

Step 2 calculates the *apex point* z by the following equation:

$$z = \tilde{x} + \left(\eta + \max\left\{\frac{b_i - \langle a_i, x'\rangle}{\langle a_i, e_c\rangle}\middle| i \in \mathcal{I}_c\right\}\right)e_c, \tag{117}$$

where $\mathcal{I}_c$ defined by equation (63) is the set of indices for which the half-space $\hat{H}_i$ is c-recessive, and $\eta \in \mathbb{R}_{>0}$ is a positive parameter. Corollary 1 guarantees that the point z chosen according to equation (117) does not belong to any c-recessive half-space $\hat{H}_i$. This choice is based on the intuition that the pseudoprojection from such a point will not be very far from the exact solution of the LP problem. The interpretation of this intuition comes from Proposition 3, which states that the solution of the LP problem (38) lies on some hyperplane H_i bounding the c-recessive half-space $\hat{H}_i$. The parameter η can significantly affect the proximity of the point $\rho_M(z)$ to the exact solution. The optimal value of η can be obtained by seeking the maximum of the objective function using the successive dichotomy method.

Step 3 calculates the initial approximation $u^{(0)}$ for the target stage by the following equation:

$$u^{(0)} = \rho_M(z). \tag{118}$$

Numerous computational experiments show that the process of calculating the pseudoprojection by Definition 4 starting from an exterior point always converges to a point on the boundary of the feasible polytope M. However, at the moment, we do not have a rigorous proof of this fact.

4.3. Target Stage

The target stage of the apex method plays the role of a corrector and calculates a sequence of points

$$\left\{u^{(0)}, u^{(1)}, \ldots, u^{(k)}, \ldots\right\} \tag{119}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

that has the following properties for all $k \in \{0, 1, 2, \ldots\}$:

$$u^{(k)} \in \Gamma_M; \tag{120}$$

$$\left\langle c, u^{(k)} \right\rangle < \left\langle c, u^{(k+1)} \right\rangle; \tag{121}$$

$$\lim_{k \to \infty} \left\| u^{(k)} - \bar{x} \right\| = 0. \tag{122}$$

Here, Γ_M stands for the set of boundary points of the feasible polytope M. Condition (120) means that all points of sequence (119) lie on the boundary of the polytope M. Condition (121) states that the value of the objective function at each next point of sequence (119) is greater than at the previous one. According to condition (122), sequence (119) converges to the exact solution of LP problem (38). An implementation of the Target stage is presented in Algorithm 3.

Algorithm 3 Target stage

Require: $\hat{H}_i = \{x \in \mathbb{R}^n | \langle a_i, x \rangle \leqslant b_i\}$, $M = \bigcap_{i=1}^{m} \hat{H}_i$, $M \neq \emptyset$

1: **input** $u^{(0)}$
2: $k := 0$
3: $v := u^{(k)} + \delta e_c$
4: $w := \rho_M(v)$
5: **while** $\left\langle c, w - u^{(k)} \right\rangle > \epsilon_f$ **do**
6: $u := u^{(k)}$
7: $d := w - u^{(k)}$
8: **while** $\|d\| > \epsilon_d$ **do**
9: **if** $(u + d) \in M$ **then**
10: $u := u + d$
11: **else**
12: $d := d/2$
13: **end if**
14: **end while**
15: $u^{(k+1)} := u$
16: $k := k + 1$
17: $v := u^{(k)} + \delta e_c$
18: $w := \rho_M(v)$
19: **end while**
20: **output** $u^{(k)}$
21: **stop**

Let us give brief comments on the steps of Algorithm 3. Step 1 reads the initial approximation $u^{(0)}$ constructed at the quest stage. Step 2 assigns zero to the iteration counter k. Step 3 adds the vector δe_c to $u^{(k)}$ and assigns the result to v. Here, e_c is a unit vector parallel to c, δ is a positive parameter. The parameter δ must be small enough to ensure that $\left\{ x \in \mathbb{R}^n \middle| x = (1 - \lambda)w - \lambda u^{(k)}, 0 \leqslant \lambda \leqslant 1 \right\} \subset \Gamma_M$. Recall that Γ_M denotes the set of boundary points of the feasible polytope M. Step 4 calculates the pseudoprojection $\rho_M(v)$ and assigns the result to w. Steps 5–19 implement the main loop. This loop is processed while the condition

$$\left\langle c, w - u^{(k)} \right\rangle > \epsilon_f \tag{123}$$

holds. Here, ϵ_f is a small positive parameter. Step 6 introduces the point u moving along the surface of the polytope M from the point $u^{(k)}$ to the next approximation $u^{(k+1)}$. Step 7 calculates the vector d, which defines the direction of movement of the point u. The loop in

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

steps 8–14 moves the point u along the surface of the polytope M in this direction as far as possible. To achieve this, the vector d is successively divided in half each time the next step moves u beyond the boundary of the polytope M. The movement stops when the length of the vector d becomes less than ϵ_d. Here, ϵ_d is a small positive parameter. Step 15 sets the next approximation $u^{(k+1)}$ using the value of u. Step 16 increases the iteration counter k by 1. Steps 17 and 18 calculate new points v and w for the next iteration of the main loop. Step 20 outputs $u^{(k)}$ as the final approximation of the exact solution $\bar{x}$ of LP problem (38). Schematically, the work of Algorithm 3 is shown in Figure 1.

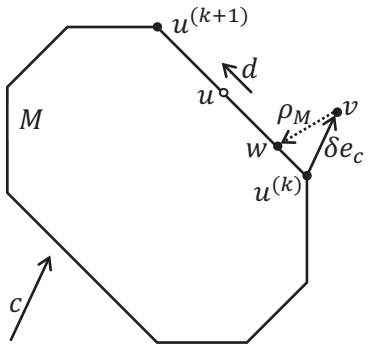

Figure 1. At each iteration of the main loop, Algorithm 3 performs the following steps. First, the algorithm constructs the vector δe_c from the point $u^{(k)}$ and obtains the point v. Second, the pseudoprojection of the point v gives the point w. Then, the direction vector d is calculated as a difference between the vectors w and $u^{(k)}$. Finally, the point u moves from the point $u^{(k)}$ in the direction d along the surface of the polytope M as far as possible. As a result, we obtain the next approximation $u^{(k+1)}$.

The following proposition guarantees the convergence of Algorithm 3.

Proposition 6. *Let the feasible polytope M of LP problem (38) be a closed bounded set, and $M \neq \emptyset$. Then, the sequence $\left\{u^{(k)}\right\}$ generated by Algorithm 3 terminates in the finite number of iteration $K \geqslant 0$, and*

$$\left\langle c, u^{(0)} \right\rangle < \left\langle c, u^{(1)} \right\rangle < \left\langle c, u^{(2)} \right\rangle < \ldots < \left\langle c, u^{(K)} \right\rangle. \tag{124}$$

Proof. The case when $K = 0$ is trivial. Let $K > 0$ or $K = \infty$. First, we show that for any $k < K$ the following inequality holds:

$$\left\langle c, u^{(k)} \right\rangle < \left\langle c, u^{(k+1)} \right\rangle. \tag{125}$$

Indeed, inequality (123) implies

$$\left\langle c, u^{(k)} \right\rangle < \langle c, w \rangle. \tag{126}$$

According to Step 7 in Algorithm 3, it follows that

$$d \neq \mathbf{0}. \tag{127}$$

Without loss of generality, we can assume that $\left\| w - u^{(k)} \right\| > \epsilon_d$. Then, according to steps 8-15, we obtain

$$u^{(k+1)} = u^{(k)} + \mu d, \tag{128}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

where $\mu > 0$. Taking into account inequality (123) and Step 7 of Algorithm 3, it follows

$$\left\langle c, u^{(k+1)} \right\rangle = \left\langle c, u^{(k)} + \mu d \right\rangle = \left\langle c, u^{(k)} + \mu \left(w - u^{(k)} \right) \right\rangle =$$
$$= \left\langle c, u^{(k)} \right\rangle + \mu \left\langle c, w - u^{(k)} \right\rangle > \left\langle c, u^{(k)} \right\rangle.$$

Now, we show that $K < \infty$. Assume the opposite, i.e. Algorithm 3 generates the infinite sequence of points. In this case, we obtain the monotonically increasing numerical sequence

$$\left\langle c, u^{(0)} \right\rangle < \left\langle c, u^{(1)} \right\rangle < \left\langle c, u^{(2)} \right\rangle < \dots \tag{129}$$

Since the feasible polytope M is bounded, sequence (129) is bounded from above. According to the monotone convergence theorem, a monotonically increasing numerical sequence bounded from above converges to its supremum. This means that there exists $K' \in \mathbb{N}$ such that

$$\forall k > K' : \left\langle c, u^{(k+1)} \right\rangle - \left\langle c, u^{(k)} \right\rangle < \epsilon_d. \tag{130}$$

It follows

$$\forall k > K' : \langle c, w \rangle - \left\langle c, u^{(k)} \right\rangle < \epsilon_d \tag{131}$$

that is equivalent to

$$\forall k > K' : \left\langle c, w - u^{(k)} \right\rangle < \epsilon_d. \tag{132}$$

Thus, we obtain a contradiction with the stopping criterion (123) used in Step 5 of Algorithm 3. □

The notion of pseudoprojection is a generalization of the notion of *metric projection*, which can be defined as follows [67].

Definition 5. *Let Q be a closed convex set in $\mathbb{R}^n$, and $Q \neq \emptyset$. The metric projection $P_Q(x)$ of the point $x \in \mathbb{R}^n$ onto the set Q is defined by the equation*

$$P_Q(x) = \arg\min\{\|x - q\| | q \in Q\}. \tag{133}$$

For the metric projection, the following proposition similar to Proposition 6 for the pseudoprojection holds.

Proposition 7. *The sequence $\left\{ u^{(k)} \right\}$ generated by Algorithm 3 with the metric projection $P_M(\cdot)$ instead of the pseudoprojection $\rho_M(\cdot)$ terminates in the finite number of iteration $K \geqslant 0$, and*

$$\left\langle c, u^{(0)} \right\rangle < \left\langle c, u^{(1)} \right\rangle < \left\langle c, u^{(2)} \right\rangle < \dots < \left\langle c, u^{(K)} \right\rangle. \tag{134}$$

Proof. The proof follows the same scheme as the proof of Proposition 6. □

The following proposition states that Algorithm 3 with metric projection converges to the exact solution of LP problem.

Proposition 8. *Let the pseudoprojection $\rho_M(\cdot)$ be replaced by the metric projection $P_M(\cdot)$ in Algorithm 3. Then, Algorithm 3 converges to the exact solution $\bar{x}$ of LP problem (38).*

Proof. Let $\bar{u}$ stand for the terminal point of the sequence $\left\{ u^{(k)} \right\}$ generated by Algorithm 3 with the metric projection $P_M(\cdot)$. This point exists according to Proposition 7. Assume the opposite, i.e., $\bar{u} \neq \bar{x}$. This is equivalent to

$$\langle c, \bar{u} \rangle < \langle c, \bar{x} \rangle. \tag{135}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

Let $S_\delta(v)$ designate the open n-ball of radius δ and center v, where

$$v = \bar{u} + \delta e_c. \tag{136}$$

According to (135), it follows that

$$S_\delta(v) \cap M \neq \emptyset. \tag{137}$$

Let

$$w = \arg\min\{\|x - v\| | x \in S_\delta(v) \cap M\}. \tag{138}$$

This is equivalent to

$$w = P_M(v). \tag{139}$$

It is easy to see that the following inequality holds:

$$\langle c, w \rangle > \langle c, \bar{u} \rangle. \tag{140}$$

Condition (136), (139), and (140) say that $\bar{u}$ is not the terminal point of the sequence $\left\{u^{(k)}\right\}$ generated by Algorithm 3. Thus, we obtain a contradiction, and the proposition is proved. □

In case of using the pseudoprojection in Algorithm 3, the convergence to the exact solution is based on the intuition that $\rho_M(v) \to P_M(v)$ with $\delta \to 0$. However, a rigorous proof of this fact is beyond the scope of this article.

5. Implementation and Computational Experiments

We implemented a parallel version of the apex method in C++ using the BSF-skeleton [73], which is based on the BSF parallel computation model [72]. The BSF-skeleton encapsulates all aspects related to the parallelization of a program using the MPI library. The source code of this implementation is freely available at https://github.com/leonid-sokolinsky/Apex-method. Using this program, we investigated the scalability of the apex method. The computational experiments were conducted on the "Tornado SUSU" computing cluster [74], whose specifications are shown in Table 1.

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

As test problems, we used random synthetic LP problems generated by the program FRaGenLP [75]. A verification of solutions obtained by the apex method was performed using the program VaLiPro [76]. We conducted a series of computational experiments in which we investigated the dependence of speedup on the number of processor nodes used. Here, the speedup α is defined as the ratio of the time $T(1)$ required by parallel algorithm using one master node and one worker node to solve a problem to the time $T(L)$ required by parallel algorithm using one master node and L worker nodes to solve the same problem:

$$\alpha = \frac{T(1)}{T(L)}. \tag{141}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

The computations were performed with the following dimensions: 5 000, 7 500 and 10 000. The number of inequalities was 10 002, 15 002 and 20 002, respectively. The results of computational experiments are presented in Figure 2.

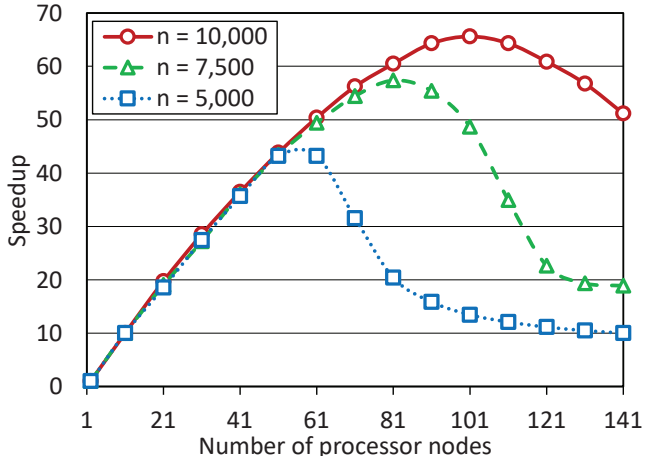

Figure 2. The curves demonstrate the dependence of speedup on the number of processor nodes used by the parallel apex algorithm. Speedup is the ratio of the time required by parallel algorithm using one master node and one worker node to solve a problem to the time required by parallel algorithm using one master node and L worker nodes to solve the same problem. Experiments show that the scalability of the parallel apex algorithm depends significantly on the dimension n of the LP problem.

These experiments showed that the scalability boundary of the parallel apex algorithm depends significantly on the size of the LP problem. At $n = 5\,000$, the scalability boundary was approximately 55 processor nodes. For the problem of dimension $n = 7\,500$, this boundary increased to 80 nodes, and for the problem of dimension $n = 10\,000$, it was close to 100 nodes. Further increasing the problem size caused the compiler error: "insufficient memory". It should be noted that the computations were performed in the double precision floating-point format occupying 64 bits in computer memory. An attempt to use the single-precision floating-point format occupying 32 bits failed because the apex algorithm stopped to converge. The experiments have also shown that the parameter η used in equation (117) to calculate the apex point z at the quest stage has a negligible effect on the total time of solving the problem when this parameter has large values (more then 100 000). If the apex point is not far enough away from the polytope, then its pseudo-projection can be an interior point of some polytope face. If the apex point is taken far enough away from the polytope (the value $\eta = 20\,000 \cdot n$ was used in the experiments), then the pseudo-projection always fell into one of the polytope vertices. Also, we would like to note that, in the case of synthetic LP problems generated by the program FRaGenLP, all computed points in the sequence $\left\{u^{(k)}\right\}$ were vertices of the polytope. The computational experiment showed that more than 99% of the time spent for solving the LP problem by the apex method was taken by the calculation of pseudoprojections (Step 18 of Algorithm 3). At that, the calculation of one approximation $u^{(k)}$ for a problem of dimension $n = 10\,000$ on 100 processor nodes took 44 minutes.

We also tested the apex method on a subset of LP problems from the Netlib-LP repository [77] available at https://netlib.org/lp/data. The Netlib suite of linear optimization problems includes many real world applications like stochastic forestry problems, oil refinery problems, flap settings of aircraft, pilot models, audit staff scheduling, truss structure problems, airline schedule planning, industrial production and allocation models, image restoration problems, and multisector economic planning problems. It contains problems ranging in size from 32 variables and 27 constraints up to 15695 variables and 16675 constraints [78]. The exact solutions (optimal values of objective functions) of all problems were obtained from paper [79]. The results are presented in Table 2.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

Table 2. Applying the apex method to the Netlib-LP problems.

No	Problem		Quest stage		Target stage	
	Name	Exact solution	Rough solution	Error	Refined solution	Error
1	adlittle	2.25494963E5	3.67140280E5	6.28E-1	2.2571324E5	9.68E-4
2	afiro	-4.64753142E2	-4.55961488E2	1.89E-2	-4.6475310E2	8.61E-9
3	blend	-3.08121498E1	-3.60232513E0	8.83E-1	-3.0811018E1	3.19E-5
4	fit1d	-9.14637809E3	-3.49931014E3	6.17E-1	-9.1463386E3	8.77E-7
5	kb2	-1.74990012E3	-1.39603193E3	2.02E-1	-1.6879152E3	3.54E-2
6	recipe	-2.66616000E2	-2.66107349E2	1.91E-3	-2.6660404E2	2.23E-5
7	sc50a	-6.45750770E1	-5.58016335E1	1.36E-1	-6.4568167E1	1.06E-4
8	sc50b	-7.00000000E1	-6.92167246E1	1.12E-2	-6.9990792E1	1.32E-4
9	sc105	-5.22020612E1	-4.28785710E1	1.79E-1	-5.1837995E1	6.97E-3
10	share2b	-4.15732240E2	-4.28792528E2	3.14E-2	-4.1572001E2	2.40E-5

These experiments showed that the relative error of the rough solution calculated at the quest stage was less than or equal to 0.2, excluding the *adlittle*, *blend*, and *fit1d* problems. The relative error of the refined solution calculated at the target stage was less than 10^{-3}, excluding the *kb2* and *sc105*, for which the error was 0.035 and 0.007, respectively. The runtime varied from a few seconds for *afiro* to tens of hours for *blend*. One of the main parameters affecting the convergence rate of the apex method was the parameter ϵ used in Step 12 of the parallel Algorithm 2 calculating the pseudoprojection. All runs are available on https://github.com/leonid-sokolinsky/Apex-method/tree/master/Runs.

6. Discussion

In this section of the article, we will discuss some issues related to the scientific contribution and applicability of the apex method in practice and give answers to the following questions.

1. What is the scientific contribution of this article?
2. What is the practical significance of the apex method?
3. What is our confidence that the apex method always converges to the exact solution of LP problem?
4. How can we speed up the convergence of the Algorithm 1 calculating a pseudoprojection on the feasible polytope M?

The main scientific contribution of this article is that it presents the apex method that allows, for the first time, as far as we know, to construct a path close to optimal on the surface of feasible polytope from a certain starting point to the exact solution of LP problem. Here, the optimal path refers to a path of minimum length according to the Euclidean metric. By intuition, moving in the direction of the greatest increase in the value of the objective function will give us the shortest path to the point of maximum of the objective function on the surface of the feasible polytope. We intend to present a formal proof of this fact in a future work.

The practical significance of the apex method is based on the issue of applying feed-forward neural networks, including convolutional neural networks, to solve LP problems. In recent paper [52], a method of visual representation of n-dimensional LP problems was proposed. This method constructs an image of the feasible polytope M in the form of a matrix I of dimension $(n-1)$ using rasterization technique. A vector antiparallel to the gradient vector of the objective function is used as the view ray. Each pixel value in I is proportional to the value of the objective function at the corresponding point on the surface of a feasible polytope M. Such an image makes it possible to use a feedforward neural network to construct the optimal path on the surface of the feasible polytope to the solution of the LP problem. Actually, the feedforward neural network can directly calculate the vector d in Algorithm 3, making the calculation of pseudoprojection redundant. The advantage of this approach is that the feed-forward neural network works in real time, which is important for robotics. We are not aware of other methods that solves LP problems

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 March 2023 doi:10.20944/preprints202303.0297.v1

in real time. However, applying a feedforward neural network to solve LP problems involves the task of preparing a training dataset. The apex method provides the possibility to construct such a training dataset.

Proposition 6 states that Algorithm 3 converges to some point on the surface of the feasible polytope M in a finite number of steps, but leaves open the question whether the terminal point will be a solution to the LP problem. According to Proposition 8, the answer to this question is positive if, in Algorithm 3, the pseudoprojection is replaced by the metric projection. However, there are no methods for constructing the metric projection for an arbitrary bounded convex polytope. Therefore, we are forced to use the pseudoprojection. Numerous experiments show that the apex method converges to the solution of the LP problem, but this fact requires a formal proof. We plan to make such a proof in our future work.

The main drawback of the apex method is the slow rate of convergence to the solution of the LP problem. The LP problem, which takes several seconds to find the optimal solution by standard linear programming solvers, may take several hours to solve by the apex method. Computational experiments demonstrated that more than 99% of the time spent for solving the LP problem by the apex method was taken by the calculation of pseudoprojections. Therefore, the issue of speeding up the process of calculating the pseudoprojection is urgent. In the apex method, the pseudoprojection is calculated by Algorithm 1, which belongs to the class of projection-type methods discussed in detail in Section 2. In the case of closed bounded polytope $M \neq \emptyset$, the projection-type methods have a low linear rate of convergence:

$$\left\| x^{(k+1)} - \rho_M\left(x^{(0)}\right) \right\| \leqslant Cq^k, \tag{142}$$

where $0 < C < \infty$ is some constant, and $q \in (0,1)$ is a parameter that depends on the angles between the half-spaces corresponding to the faces of the polytope M [80]. This means that the distance between adjacent approximations decreases at each iteration by a constant factor less than 1. For small angles, the convergence rate can decrease to values close to zero. This fundamental limitation of the projection-type methods cannot be overcome. However, we can reduce the number of half-spaces used to compute the pseudoprojection. According to Proposition 3, the solution of LP problem (38) belongs to some c-recessive half-space. Hence, in Algorithm 1 calculating the pseudoprojection, we can take into account only the c-recessive hyperplanes. On average, this reduces the number of half-spaces by two times. Another way to reduce the pseudoprojection calculation time is to parallelize Algorithm 1, as was done in Algorithm 2. However, the degree of parallelism in this case will be limited by the theoretical estimation (115).

7. Conclusion

In this paper, we proposed a new scalable iterative method for linear programming called the "apex method". The key feature of this method is constructing a path close to optimal on the surface of the feasible region from a certain starting point to the exact solution of linear programming problem. The optimal path refers to a path of minimum length according to the Euclidean metric. The main practical contribution of the apex method is that it opens the possibility of using feedforward neural networks to solve multidimensional LP problems.

The paper presents a theoretical basis used to construct the apex method. The half-spaces generated by the constraints of the LP problem are considered. These half-spaces form the feasible polytope M, which is a closed bounded set. These half-spaces are divided into two groups with respect to the gradient c of the linear objective function: c-dominant and c-recessive. The necessary and sufficient condition for the c-recessivity is obtained. It is proved that the solution of the LP problem lies on the boundary of a c-recessive half-space. The equation defining the apex point not belonging to any c-recessive half-space is derived. The apex point is used to calculate the initial approximation on the surface

of the feasible polytope M. The apex method constructs a path close to optimal on the surface of the feasible polytope M from this initial approximation to the solution of the LP problem. To do this, it uses a parallel algorithm constructing the pseudoprojection, which is a generalization of the notion of metric projection. For this parallel algorithm an analytical estimation of the scalability bound is obtained. This estimation says that the scalability boundary of the parallel algorithm of calculating the pseudoprojection on a cluster computing system does not exceed $O\left(\sqrt{m}\right)$ processor nodes, where m is the number of constrains of the linear programming problem. The algorithm constructing a path close to optimal on the surface of the feasible polytope, from the initial approximation to the exact solution of linear programming problem, is described. The convergence of this algorithm is proven.

The parallel version of the apex method is implemented in C++ using the BSF-skeleton based on the BSF parallel computation model. Large-scale computational experiments were conducted to investigate the scalability of the apex method on a cluster computing system. These experiments show that, for a synthetic scalable linear programming problem with a dimension of 10 000 and a constraint number of 20 002, the scalability boundary of the apex methods is close to 100 processor nodes. At the same time, these experiments showed that more than 99% of the time spent for solving the LP problem by the apex method was taken by the calculation of pseudoprojections.

In addition, the apex method was used to solve 10 problems from the Netlib-LP repository. These experiments showed that the relative error varied from $3.5 \cdot 10^{-3}$ to $8.6 \cdot 10^{-9}$. The runtime ranged from a few seconds for to tens of hours. The main parameter affecting the convergence rate of the apex method was the precision of calculating the pseudoprojection.

Our future research directions on this subject are as following. We plan to develop a new method for calculating the pseudoprojection onto the feasible polytope of the LP problem. The basic idea is to reduce the number of half-spaces used in one iteration. At the same time, the number of these half-spaces should remain large enough to enable the efficient parallelization. The new method should outperform Algorithm 2 in terms of convergence rate. We will also need to prove that the new method converges to a point that lies on the boundary of the feasible region. In addition, we plan to investigate the usefulness of utilizing the linear superiorization technique [70] in the apex method.

Author Contributions: All authors contributed equally to the main text. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by RSF (project No. 23-21-00356).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sokolinsky, L.B.; Sokolinskaya, I.M. Scalable Method for Linear Optimization of Industrial Processes. In Proceedings of the Proceedings - 2020 Global Smart Industry Conference, GloSIC 2020. IEEE, 2020, pp. 20–26. Article number 9267854. https://doi.org/10.1109/GloSIC50886.2020.9267854.
2. Jagadish, H.V.; Gehrke, J.; Labrinidis, A.; Papakonstantinou, Y.; Patel, J.M.; Ramakrishnan, R.; Shahabi, C. Big data and its technical challenges. *Communications of the ACM* **2014**, *57*, 86–94. https://doi.org/10.1145/2611567.
3. Hartung, T. Making Big Sense From Big Data. *Frontiers in Big Data* **2018**, *1*, 5. https://doi.org/10.3389/fdata.2018.00005.
4. Sokolinskaya, I.; Sokolinsky, L.B. On the Solution of Linear Programming Problems in the Age of Big Data. In Proceedings of the Parallel Computational Technologies. PCT 2017. Communications in Computer and Information Science, vol. 753.; Sokolinsky, L.; Zymbler, M., Eds.; Springer: Cham, 2017; pp. 86–100. https://doi.org/10.1007/978-3-319-67035-5_7.
5. Chung, W. Applying large-scale linear programming in business analytics. In Proceedings of the 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM). IEEE, 2015, pp. 1860–1864. https://doi.org/10.1109/IEEM.2015.7385970.

6. Gondzio, J.; Gruca, J.A.; Hall, J.A.J.; Laskowski, W.; Zukowski, M. Solving large-scale optimization problems related to Bell's Theorem. *Journal of Computational and Applied Mathematics* **2014**, *263*, 392–404. https://doi.org/10.1016/j.cam.2013.12.003.
7. Sodhi, M.S. LP modeling for asset-liability management: A survey of choices and simplifications. *Operations Research* **2005**, *53*, 181–196. https://doi.org/10.1287/opre.1040.0185.
8. Branke, J. Optimization in Dynamic Environments. In *Evolutionary Optimization in Dynamic Environments. Genetic Algorithms and Evolutionary Computation, vol. 3*; Springer: Boston, MA, 2002; pp. 13–29. https://doi.org/10.1007/978-1-4615-0911-0_2.
9. Brogaard, J.; Hendershott, T.; Riordan, R. High-Frequency Trading and Price Discovery. *Review of Financial Studies* **2014**, *27*, 2267–2306. https://doi.org/10.1093/rfs/hhu032.
10. Deng, S.; Huang, X.; Wang, J.; Qin, Z.; Fu, Z.; Wang, A.; Yang, T. A Decision Support System for Trading in Apple Futures Market Using Predictions Fusion. *IEEE Access* **2021**, *9*, 1271–1285. https://doi.org/10.1109/ACCESS.2020.3047138.
11. Seregin, G. *Lecture notes on regularity theory for the Navier-Stokes equations*; World Scientific Publishing Company: Singapore, 2014; p. 268. https://doi.org/10.1142/9314.
12. Demin, D.A. Synthesis of optimal control of technological processes based on a multialternative parametric description of the final state. *Eastern-European Journal of Enterprise Technologies* **2017**, *3*, 51–63. https://doi.org/10.15587/1729-4061.2017.105294.
13. Kazarinov, L.S.; Shnayder, D.A.; Kolesnikova, O.V. Heat load control in steam boilers. In Proceedings of the 2017 International Conference on Industrial Engineering, Applications and Manufacturing, ICIEAM 2017 - Proceedings. IEEE, 2017. https://doi.org/10.1109/ICIEAM.2017.8076177.
14. Zagoskina, E.V.; Barbasova, T.A.; Shnaider, D.A. Intelligent Control System of Blast-furnace Melting Efficiency. In Proceedings of the SIBIRCON 2019 - International Multi-Conference on Engineering, Computer and Information Sciences, Proceedings. IEEE, 2019, pp. 710–713. https://doi.org/10.1109/SIBIRCON48586.2019.8958221.
15. Fleming, J.; Yan, X.; Allison, C.; Stanton, N.; Lot, R. Real-time predictive eco-driving assistance considering road geometry and long-range radar measurements. *IET Intelligent Transport Systems* **2021**, *15*, 573–583. https://doi.org/10.1049/ITR2.12047.
16. Scholl, M.; Minnerup, K.; Reiter, C.; Bernhardt, B.; Weisbrodt, E.; Newiger, S. Optimization of a thermal management system for battery electric vehicles. In Proceedings of the 14th International Conference on Ecological Vehicles and Renewable Energies, EVER 2019. IEEE, 2019. https://doi.org/10.1109/EVER.2019.8813657.
17. Meisel, S. Dynamic Vehicle Routing. In *Anticipatory Optimization for Dynamic Decision Making. Operations Research/Computer Science Interfaces Series, vol. 51*; Springer: New York, NY, 2011; pp. 77–96. https://doi.org/10.1007/978-1-4614-0505-4_6.
18. Cheng, A.M.K. Real-Time Scheduling and Schedulability Analysis. In *Real-Time Systems: Scheduling, Analysis, and Verification*; John Wiley and Sons, 2002; pp. 41–85. https://doi.org/10.1002/0471224626.CH3.
19. Kopetz, H. Real-Time Scheduling. In *Real-Time Systems. Real-Time Systems Series*; Springer: Boston, MA, 2011; pp. 239–258. https://doi.org/10.1007/978-1-4419-8237-7_10.
20. Dantzig, G.B. *Linear programming and extensions*; Princeton university press: Princeton, N.J., 1998; p. 656.
21. Hall, J.; McKinnon, K. Hyper-sparsity in the revised simplex method and how to exploit it. *Computational Optimization and Applications* **2005**, *32*, 259–283. https://doi.org/10.1007/s10589-005-4802-0.
22. Klee, V.; Minty, G. How good is the simplex algorithm? In Proceedings of the Inequalities - III. Proceedings of the Third Symposium on Inequalities Held at the University of California, Los Angeles, Sept. 1-9, 1969; Shisha, O., Ed.; Academic Press: New York-London, 1972; pp. 159–175.
23. Jeroslow, R. The simplex algorithm with the pivot rule of maximizing criterion improvement. *Discrete Mathematics* **1973**, *4*, 367–377. https://doi.org/10.1016/0012-365X(73)90171-4.
24. Zadeh, N. A bad network problem for the simplex method and other minimum cost flow algorithms. *Mathematical Programming* **1973**, *5*, 255–266. https://doi.org/10.1007/BF01580132.
25. Bartels, R.; Stoer, J.; Zenger, C. A Realization of the Simplex Method Based on Triangular Decompositions. In *Handbook for Automatic Computation. Volume II: Linear Algebra*; Springer: Berlin, Heidelberg, 1971; pp. 152–190. https://doi.org/10.1007/978-3-642-86940-2_11.
26. Tolla, P. A Survey of Some Linear Programming Methods. In *Concepts of Combinatorial Optimization*, 2 ed.; Paschos, V.T., Ed.; John Wiley and Sons: Hoboken, NJ, USA, 2014; chapter 7, pp. 157–188. https://doi.org/10.1002/9781119005216.ch7.

27. Hall, J. Towards a practical parallelisation of the simplex method. *Computational Management Science* **2010**, *7*, 139–170. https://doi.org/10.1007/s10287-008-0080-5.
28. Mamalis, B.; Pantziou, G. Advances in the Parallelization of the Simplex Method. In *Algorithms, Probability, Networks, and Games. Lecture Notes in Computer Science, vol. 9295*; Zaroliagis, C.; Pantziou, G.; Kontogiannis, S., Eds.; Springer: Cham, 2015; pp. 281–307. https://doi.org/10.1007/978-3-319-24024-4_17.
29. Lubin, M.; Hall, J.A.J.; Petra, C.G.; Anitescu, M. Parallel distributed-memory simplex for large-scale stochastic LP problems. *Computational Optimization and Applications* **2013**, *55*, 571–596. https://doi.org/10.1007/s10589-013-9542-y.
30. Khachiyan, L. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics* **1980**, *20*, 53–72. https://doi.org/10.1016/0041-5553(80)90061-0.
31. Shor, N.Z. Cut-off method with space extension in convex programming problems. *Cybernetics and Systems Analysis* **1977**, *13*, 94–96. https://doi.org/10.1007/BF01071394.
32. Nesterov, Y.; Nemirovskii, A. *Interior-Point Polynomial Algorithms in Convex Programming*; Society for Industrial and Applied Mathematics, 1994; pp. ix + 396. https://doi.org/10.1137/1.9781611970791.
33. Karmarkar, N. A new polynomial-time algorithm for linear programming. *Combinatorica* **1984**, *4*, 373–395. https://doi.org/10.1007/BF02579150.
34. Gondzio, J. Interior point methods 25 years later. *European Journal of Operational Research* **2012**, *218*, 587–601. https://doi.org/10.1016/j.ejor.2011.09.017.
35. Fathi-Hafshejani, S.; Mansouri, H.; Reza Peyghami, M.; Chen, S. Primal–dual interior-point method for linear optimization based on a kernel function with trigonometric growth term. *Optimization* **2018**, *67*, 1605–1630. https://doi.org/10.1080/02331934.2018.1482297.
36. Asadi, S.; Mansouri, H. A Mehrotra type predictor-corrector interior-point algorithm for linear programming. *Numerical Algebra, Control and Optimization* **2019**, *9*, 147–156. https://doi.org/10.3934/naco.2019011.
37. Yuan, Y. Implementation tricks of interior-point methods for large-scale linear programs. In Proceedings of the Proc. SPIE, vol. 8285. International Conference on Graphic and Image Processing (ICGIP 2011). International Society for Optics and Photonics, 2011. https://doi.org/10.1117/12.913019.
38. Kheirfam, B.; Haghighi, M. A full-Newton step infeasible interior-point method for linear optimization based on a trigonometric kernel function. *Optimization* **2016**, *65*, 841–857. https://doi.org/10.1080/02331934.2015.1080255.
39. Xu, Y.; Zhang, L.; Zhang, J. A full-modified-newton step infeasible interior-point algorithm for linear optimization. *Journal of Industrial and Management Optimization* **2016**, *12*, 103–116. https://doi.org/10.3934/jimo.2016.12.103.
40. Roos, C.; Terlaky, T.; Vial, J.P. *Interior Point Methods for Linear Optimization*; Springer: New York, 2005; p. 500. https://doi.org/10.1007/b100325.
41. Sokolinskaya, I. Parallel Method of Pseudoprojection for Linear Inequalities. In *Parallel Computational Technologies. PCT 2018. Communications in Computer and Information Science, vol. 910*; Sokolinsky, L.; Zymbler, M., Eds.; Springer: Cham, 2018; pp. 216–231. https://doi.org/10.1007/978-3-319-99673-8_16.
42. Eremin, I.I. Methods of Fejer's approximations in convex programming. *Mathematical Notes of the Academy of Sciences of the USSR* **1968**, *3*, 139–149. https://doi.org/10.1007/BF01094336.
43. Gondzio, J.; Grothey, A. Direct Solution of Linear Systems of Size 109 Arising in Optimization with Interior Point Methods. In Proceedings of the Parallel Processing and Applied Mathematics. PPAM 2005. Lecture Notes in Computer Science, vol. 3911; Wyrzykowski, R.; Dongarra, J.; Meyer, N.; Wasniewski, J., Eds.; Springer: Berlin, Heidelberg, 2006; Vol. 3911 LNCS, pp. 513–525. https://doi.org/10.1007/11752578_62.
44. Prieto, A.; Prieto, B.; Ortigosa, E.M.; Ros, E.; Pelayo, F.; Ortega, J.; Rojas, I. Neural networks: An overview of early research, current frameworks and new challenges. *Neurocomputing* **2016**, *214*, 242–268. https://doi.org/10.1016/j.neucom.2016.06.014.
45. Raina, R.; Madhavan, A.; Ng, A.Y. Large-scale deep unsupervised learning using graphics processors. In Proceedings of the Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09); ACM Press: New York, NY, USA, 2009; pp. 873–880. https://doi.org/10.1145/1553374.1553486.
46. Tank, D.W.; Hopfield, J.J. Simple 'neural' optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit. *IEEE transactions on circuits and systems* **1986**, *CAS-33*, 533–541. https://doi.org/10.1109/TCS.1986.1085953.

47. Kennedy, M.P.; Chua, L.O. Unifying the Tank and Hopfield Linear Programming Circuit and the Canonical Nonlinear Programming Circuit of Chua and Lin. *IEEE Transactions on Circuits and Systems* **1987**, *34*, 210–214. https://doi.org/10.1109/TCS.1987.1086095.
48. Rodriguez-Vazquez, A.; Dominguez-Castro, R.; Rueda, A.; Huertas, J.L.; Sanchez-Sinencio, E. Nonlinear Switched-Capacitor "Neural" Networks for Optimization Problems. *IEEE Transactions on Circuits and Systems* **1990**, *37*, 384–398. https://doi.org/10.1109/31.52732.
49. Zak, S.H.; Upatising, V. Solving Linear Programming Problems with Neural Networks: A Comparative Study. *IEEE Transactions on Neural Networks* **1995**, *6*, 94–104. https://doi.org/10.1109/72.363446.
50. Malek, A.; Yari, A. Primal–dual solution for the linear programming problems using neural networks. *Applied Mathematics and Computation* **2005**, *167*, 198–211. https://doi.org/10.1016/J.AMC.2004.06.081.
51. Liu, X.; Zhou, M. A one-layer recurrent neural network for non-smooth convex optimization subject to linear inequality constraints. *Chaos, Solitons and Fractals* **2016**, *87*, 39–46. https://doi.org/10.1016/j.chaos.2016.03.009.
52. Olkhovsky, N.; Sokolinsky, L. Visualizing Multidimensional Linear Programming Problems. In Proceedings of the Parallel Computational Technologies. PCT 2022. Communications in Computer and Information Science, vol. 1618; Sokolinsky, L.; Zymbler, M., Eds.; Springer: Cham, 2022; pp. 172–196. https://doi.org/10.1007/978-3-031-11623-0_13.
53. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. https://doi.org/10.1038/nature14539.
54. Lachhwani, K. Application of Neural Network Models for Mathematical Programming Problems: A State of Art Review. *Archives of Computational Methods in Engineering* **2020**, *27*, 171–182. https://doi.org/10.1007/s11831-018-09309-5.
55. Kaczmarz, S. Angenherte Auflsung von Systemen linearer Gleichungen. *Bulletin International de l'Acadmie Polonaise des Sciences et des Lettres. Classe des Sciences Mathmatiques et Naturelles. Srie A, Sciences Mathmatiques* **1937**, *35*, 355–357.
56. Kaczmarz, S. Approximate solution of systems of linear equations. *International Journal of Control* **1993**, *57*, 1269–1271. https://doi.org/10.1080/00207179308934446.
57. Cimmino, G. Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari. *La Ricerca Scientifica, XVI, Series II, Anno IX, 1* **1938**, pp. 326–333.
58. Gastinel, N. *Linear Numerical Analysis*; Academic Press: New York, 1971; pp. ix+341.
59. Agmon, S. The relaxation method for linear inequalities. *Canadian Journal of Mathematics* **1954**, *6*, 382–392. https://doi.org/10.4153/CJM-1954-037-2.
60. Motzkin, T.S.; Schoenberg, I.J. The relaxation method for linear inequalities. *Canadian Journal of Mathematics* **1954**, *6*, 393–404. https://doi.org/10.4153/CJM-1954-038-x.
61. Censor, Y.; Elfving, T. New methods for linear inequalities. *Linear Algebra and its Applications* **1982**, *42*, 199–211. https://doi.org/10.1016/0024-3795(82)90149-5.
62. De Pierro, A.R.; Iusem, A.N. A simultaneous projections method for linear inequalities. *Linear Algebra and its Applications* **1985**, *64*, 243–253. https://doi.org/10.1016/0024-3795(85)90280-0.
63. Sokolinskaya, I.; Sokolinsky, L. Revised Pursuit Algorithm for Solving Non-stationary Linear Programming Problems on Modern Computing Clusters with Manycore Accelerators. In *Supercomputing. RuSCDays 2016. Communications in Computer and Information Science*; Voevodin, V.; Sobolev, S., Eds.; Springer: Cham, 2016; Vol. 687, pp. 212–223. https://doi.org/10.1007/978-3-319-55669-7_17.
64. Sokolinskaya, I.M.; Sokolinsky, L.B. Scalability Evaluation of Cimmino Algorithm for Solving Linear Inequality Systems on Multiprocessors with Distributed Memory. *Supercomputing Frontiers and Innovations* **2018**, *5*, 11–22. https://doi.org/10.14529/jsfi180202.
65. Sokolinsky, L.B.; Sokolinskaya, I.M. Scalable parallel algorithm for solving non-stationary systems of linear inequalities. *Lobachevskii Journal of Mathematics* **2020**, *41*, 1571–1580. https://doi.org/10.1134/S1995080220080181.
66. Gonzalez-Gutierrez, E.; Todorov, M.I. A relaxation method for solving systems with infinitely many linear inequalities. *Optimization Letters* **2012**, *6*, 291–298. https://doi.org/10.1007/s11590-010-0244-4.
67. Vasin, V.V.; Eremin, I.I. *Operators and Iterative Processes of Fejer Type. Theory and Applications*; Inverse and III-Posed Problems Series, Walter de Gruyter: Berlin, New York, 2009; p. 155. https://doi.org/10.1515/9783110218190.
68. Eremin, I.I.; Popov, L.D. Fejer processes in theory and practice: Recent results. *Russian Mathematics* **2009**, *53*, 36–55. https://doi.org/10.3103/S1066369X09010022.

69. Nurminski, E.A. Single-projection procedure for linear optimization. *Journal of Global Optimization* **2016**, *66*, 95–110. https://doi.org/10.1007/S10898-015-0337-9.
70. Censor, Y. Can linear superiorization be useful for linear optimization problems? *Inverse Problems* **2017**, *33*, 044006. https://doi.org/10.1088/1361-6420/33/4/044006.
71. Gould, N.I. How good are projection methods for convex feasibility problems? *Computational Optimization and Applications* **2008**, *40*, 1–12. https://doi.org/10.1007/S10589-007-9073-5.
72. Sokolinsky, L.B. BSF: A parallel computation model for scalability estimation of iterative numerical algorithms on cluster computing systems. *Journal of Parallel and Distributed Computing* **2021**, *149*, 193–206. https://doi.org/10.1016/j.jpdc.2020.12.009.
73. Sokolinsky, L.B. BSF-skeleton: A Template for Parallelization of Iterative Numerical Algorithms on Cluster Computing Systems. *MethodsX* **2021**, *8*, Article number 101437. https://doi.org/10.1016/j.mex.2021.101437.
74. Dolganina, N.; Ivanova, E.; Bilenko, R.; Rekachinsky, A. HPC Resources of South Ural State University. In Proceedings of the Parallel Computational Technologies. PCT 2022. Communications in Computer and Information Science, vol. 1618; Sokolinsky, L.; Zymbler, M., Eds.; Springer: Cham, 2022; pp. 43–55. https://doi.org/10.1007/978-3-031-11623-0_4.
75. Sokolinsky, L.B.; Sokolinskaya, I.M. FRaGenLP: A Generator of Random Linear Programming Problems for Cluster Computing Systems. In Proceedings of the Parallel Computational Technologies. PCT 2021. Communications in Computer and Information Science, vol. 1437; Sokolinsky, L.; Zymbler, M., Eds.; Springer: Cham, 2021; pp. 164–177. https://doi.org/10.1007/978-3-030-81691-9_12.
76. Sokolinsky, L.B.; Sokolinskaya, I.M. VaLiPro: Linear Programming Validator for Cluster Computing Systems. *Supercomputing Frontiers and Innovations* **2021**, *8*, 51–61. https://doi.org/10.14529/js210303.
77. Gay, D.M. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Bulletin* **1985**, pp. 10–12.
78. Keil, C.; Jansson, C. Computational experience with rigorous error bounds for the netlib linear programming library. *Reliable Computing* **2006**, *12*, 303–321. https://doi.org/10.1007/S11155-006-9004-7/METRICS.
79. Koch, T. The final NETLIB-LP results. *Operations Research Letters* **2004**, *32*, 138–142. https://doi.org/10.1016/S0167-6377(03)00094-4.
80. Deutsch, F.; Hundal, H. The rate of convergence for the cyclic projections algorithm I: Angles between convex sets. *Journal of Approximation Theory* **2006**, *142*, 36–55. https://doi.org/10.1016/J.JAT.2006.02.005.