Can Linear Superiorization Be Useful for Linear Optimization Problems?

Yair Censor
Department of Mathematics
University of Haifa
Mt. Carmel, Haifa 3498838, Israel
(yair@math.haifa.ac.il)

March 17, 2016. Revised: July 19, 2016. Revised: September 27, 2016.

Abstract

Linear superiorization considers linear programming problems but instead of attempting to solve them with linear optimization methods it employs perturbation resilient feasibility-seeking algorithms and steers them toward reduced (not necessarily minimal) target function values. The two questions that we set out to explore experimentally are (i) Does linear superiorization provide a feasible point whose linear target function value is lower than that obtained by running the same feasibility-seeking algorithm without superiorization under identical conditions? and (ii) How does linear superiorization fare in comparison with the Simplex method for solving linear programming problems? Based on our computational experiments presented here, the answers to these two questions are: "yes" and "very well", respectively.

Keywords: Superiorization, bounded perturbation resilience, linear superiorization, linear programming, Simplex algorithm, feasibility-seeking, algorithmic operator, Agmon-Motzkin-Schoenberg algorithm, linear inequalities, linear feasibility problem.

1 Introduction

In this paper we propose the linear superiorization method as a tool for handling linear programming problems. The linear superiorization method is not guaranteed to find a minimum point of the linear optimization problem but it steers the linear feasibility-seeking algorithm that it employs toward points with reduced target function value. This task is not identical with that of finding a minimizer to the linear programming problem but for huge sized problems, it puts in the hands of the end-user a viable alternative to the Simplex method of linear programming, against which we compared it here.

The paper relies on previous theoretical work about superiorization included in the papers cited in the sequel, in particular [18]. Our working tools here are only experimental computations. In spite of this, we are not discussing computational issues per se but using computations as a tool in an exploratory practical validation, style "proof of concept"[1].

What is superiorization? Many constrained optimization methods are based on methods for unconstrained optimization that are adapted to deal with constraints. Such is, for example, the class of projected gradient methods wherein the unconstrained minimization inner step "leads" the process and a projection onto the whole constraints set (the feasible set) is performed after each minimization step in order to regain feasibility. This projection onto the constraints set is in itself a non-trivial optimization problem and the need to solve it in every iteration hinders the projected gradient methods and limits their efficiency to only feasible sets that are "simple to project on". Barrier or penalty methods likewise are based on unconstrained optimization married with various "add-on"s that guarantee that the constraints are preserved. Regularization methods embed the constraints into the objective function and proceed with unconstrained solution methods for the "regularized" new objective function.

In contrast to these approaches, the superiorization methodology can be viewed as an antipodal way of thinking. Instead of adapting unconstrained minimization algorithms to handling constraints it adapts feasibility-seeking

[1] "A proof of concept (POC) or a proof of principle is a realization of a certain method or idea to demonstrate its feasibility, or a demonstration in principle, whose purpose is to verify that some concept or theory has the potential of being used. A proof of concept is usually small and may or may not be complete". (https://en.wikipedia.org/wiki/Proof_of_concept).

algorithms to reduce target function values. This is done while retaining the feasibility-seeking nature of the algorithm, and without paying a high "computational price".

Usefulness of the approach. The usefulness of this approach relies on two features: (i) **Computational**: feasibility-seeking is logically a less-demanding task than seeking a constrained minimization point in a feasible set. Therefore, letting efficient feasibility-seeking algorithms "lead" the algorithmic effort and modifying them with inexpensive add-ons works well in practice. (ii) **Applicational**: in some significant real-world applications the choice of a target function is exogenous to the modeling and data collection which give rise to the constraints. In such situations the limited confidence in the usefulness of a chosen target function leads often to the recognition that, from the application-at-hand point of view, there is no need, neither a justification, to search for an exact constrained minimum. For obtaining "good results", evaluated by how well they serve the task of the application at hand, it is often enough to find a feasible point that has reduced (not necessarily minimal) target function value. In some operations research applications, the target functions are costs or profits and are central to the model but in others the above reasoning may still apply[2].

Current research. Current work on superiorization can be appreciated from the materials on the Internet page [10]. In particular, [26] and [9] are reviews of interest. Recent research includes a variety of reports ranging from new applications in industrial x-ray computed tomography [39] to new mathematical results on the foundation of superiorization such as strict Fejér monotonicity by superiorization of feasibility-seeking projection methods [18]. A recent detailed description of previous work related to superiorization can be found in [15, Section 3].

Linear superiorization. Linear superiorization (henceforth abbreviated: LinSup) considers linear programming (LP) problems wherein the constraints as well as the objective function are linear. The two questions that we set out to explore experimentally here are: (i) Does LinSup provide a feasible point whose target function value is lower than that obtained by running the

[2]Some support for this reasoning may be borrowed from the American scientist and Noble-laureate Herbert Simon who was in favor of "satisficing" rather than "maximizing". Satisficing is a decision-making strategy that aims for a satisfactory or adequate result, rather than the optimal solution. This is because aiming for the optimal solution may necessitate needless expenditure of time, energy and resources. The term "satisfice" was coined by Herbert Simon in 1956 [40], see: https://en.wikipedia.org/wiki/Satisficing.

same feasibility-seeking algorithm without superiorization but under otherwise identical conditions? and (ii) How does LinSup fare in comparison with the Simplex method for solving LP problems? Based on our computational experiments presented here, the answers to these two questions are: "yes" and "very well", respectively.

An interesting and promising aspect of the current experiments is the dependence of the results on the test problem sizes. We found that the advantages of LinSup become monotonically more pronounced as the problem sizes increase. We treated problems of up to $8,000$ linear inequalities and vectors of up to $10,000$ components, but the trend is visible and if it persists beyond these problem sizes then LinSup might well become a useful computational tool for huge size problems. Admittedly, our preliminary work presented here relies on randomly generated problems and these are not typical of the kinds of problems that linear programming has been called to solve over the years.

We show in Section 4 that LinSup finds a superior feasible point, i.e., a feasible point with lower target function value. In Section 5 we demonstrate the computational behavior of LinSup versus the classical Simplex algorithm for linear optimization. The general framework of superiorization appears in Section 2 and LinSup is then presented in Section 3. Our experimental results were generated with MATLAB [31] and are presented in Sections 4 and 5. We make concluding remarks in Section 6 and list a variety of questions for further research on LinSup. The Appendix (Section 7) briefly describes the technical changes and modifications that the algorithmic structure of the superiorized version of a basic algorithm has undergone in the published literature over the past several years since it inception.

2 The superiorization methodology

Consider a pair $(M, \mathcal{A})$ where M, called a target set, is a given subset of a given subset Q of the J-dimensional Euclidean space $M \subset Q \subseteq R^J$. Let $\mathcal{A} : Q \rightarrow Q$ be an algorithmic operator that defines an iterative process, called the basic algorithm[3],

$$x^0 \in Q,\ x^{k+1} = \mathcal{A}(x^k),\ k = 1, 2, \ldots \tag{1}$$

[3]It will become truly an algorithm after a stopping rule will be added to it.

whose task is to find a point in the target set M. We, henceforth, refer to such a pair $(M, \mathcal{A})$ as a superiorization pair. Let $\phi : Q \subseteq R^J \rightarrow R$ be a given real-valued function, called a target function

The superiorization methodology is intended for constrained function reduction problems of the following form.

Problem 1 ***The Constrained Function Reduction Problem****. Let $(M, \mathcal{A})$ be a superiorization pair and let $\phi : Q \subseteq R^J \rightarrow R$ be a target function. Find a point x^* of M whose function ϕ value is less (but not necessarily minimal) than that of a point in M that would have been reached by applying the basic algorithm for finding a point of M.*

The superiorization methodology approaches this problem by investigating the perturbation resilience of the basic algorithm, and then using proactively such perturbations in order to "force" the perturbed algorithm obtained from the basic algorithm to do, in addition to its original task, also target function reduction steps. The so perturbed algorithm is called "the superiorized version of the basic algorithm".

If the basic algorithm is computationally efficient and useful, in terms of an application at hand, for finding a point of M and if it is perturbation resilient and the perturbations are simple and not expensive to calculate, then the advantage of this method is that, for essentially the computational cost of the basic algorithm, we are able to solve the constrained function reduction problem by steering the iterates according to the target function reduction perturbations. The superiorization methodology automatically generates the superiorized version of the basic algorithm. The vector x^*, obtained by applying the superiorized version of the basic algorithm, need not be a minimizer of ϕ over M. For further details about the kinds of perturbation resilience that may be used consult, e.g., [9, Definitions 4 and 9] or [13, 15, 27].

The above definitions and terminology depend on what precise meaning we attach to the statement "Find a point x^* of M" in Problem 1. In weak superiorization, "finding a point of M" is understood as generating an infinite sequence $\{x^k\}_{k=0}^{\infty}$ that converges to a point $x^* \in M$, thus M must be nonempty. In strong superiorization "finding a point of M" is understood as finding a point x^* that is ε-compatible with M, for some positive ε, i.e., a point whose proximity function which measures by how much it violates M has value smaller or equal to ε. Thus, nonemptiness of M need not be assumed. These notions were defined in [9].

Two significant special cases of superiorization pairs $(M, \mathcal{A})$ in the above framework come to mind although other cases are also possible.

Case 2 *The target set M is the solution set of a convex feasibility problem (CFP) of the form: Find a vector $x^* \in \cap_{i=1}^{I} C_i$, where $C_i \subseteq R^J$ are closed convex subsets, thus $M = \cap_{i=1}^{I} C_i$. In this case the algorithmic operator and the basic algorithm (1) it entails can be any of the wide variety of feasibility-seeking algorithms, see, e.g., [3, 4, 8, 11, 12, 19].*

Case 3 *The target set M is the solution set of another constrained minimization problem:* minimize $\{f(x) \mid x \in \Omega\}$ *of an objective function f over a feasible region Ω, thus $M := \{x^* \in \Omega \mid f(x^*) \leq f(x)$ for all $x \in \Omega\}$. In this case the algorithmic operator and the basic algorithm (1) it entails can be any of the wide variety of constrained minimization algorithms abundant in the literature.*

In this paper we do linear superiorization which concentrates on the special situation of Case 2 wherein all constraint sets C_i as well as the target function ϕ are linear. Superiorization work with other target functions such as total variation (TV) appears in, e.g., [13, 15, 27]. Superiorization work on Case 3, where M is the solution set of a maximum likelihood optimization problem appears in [23, 28, 30].

3 Linear superiorization

3.1 The problem and the algorithm

Let the target set M be

$$M := \{x \in R^J \mid Ax \leq b, \ x \geq 0\} \tag{2}$$

where the $I \times J$ real matrix $A = (a_j^i)_{i=1,j=1}^{I,J}$ and the vector $b = (b_i)_{i=1}^{I} \in R^I$ are given.

For a basic algorithm we pick a feasibility-seeking projection method. Projections onto sets are used in many methods in optimization theory but here projection methods refer to iterative algorithms that use projections onto sets while relying on the general principle that when a family of, usually closed and convex, sets is present, then projections onto the individual sets are easier to perform than projections onto other sets (intersections, image sets under some transformation, etc.) that are derived from the individual sets.

Projection methods may have different algorithmic structures, such as block-iterative projections (BIP), see, e.g., [22, 24] and references therein, or string-averaging projections (SAP), see, e.g., [17] and references therein, of which some are particularly suitable for parallel computing, and they demonstrate nice convergence properties and/or good initial behavior patterns. This class of algorithms has witnessed great progress in recent years and its member algorithms have been applied with success to many scientific, technological and mathematical problems. See, e.g., the 1996 review [3], the recent annotated bibliography of books and reviews [11] and its references, the excellent book [8], or [12].

An important comment is in place here. A convex feasibility problem, mentioned in Case 2, can be translated into an unconstrained minimization of some proximity function that measures the feasibility violation of points. For example, using a weighted sum of squares of the Euclidean distances to the sets of the CFP as a proximity function and applying steepest descent to it results in a simultaneous projections method for the CFP of the Cimmino type. However, there is no proximity function that would yield the sequential projections method for CFPs of the Kaczmarz type, see [2]. Therefore, the study of feasibility-seeking algorithms for the CFP has developed independently of minimization methods and it still vigorously does, see the references mentioned above. Over the years researchers have tried to harness projection methods for the convex feasibility problem to LP in more than one way, see, e.g., Chinneck's book [20]. The mini-review of relations between linear programming and feasibility-seeking algorithms in [34, Section 1] sheds more light on this. Our results lead us to wonder whether LinSup can serve such a cause.

The target function for linear superiorization will be

$$\phi(x) := \langle c, x \rangle \tag{3}$$

where $\langle c, x \rangle$ is the inner product of x and a given $c \in R^J$.

In the footsteps of the general principles of the superiorization methodology, as presented for general target functions ϕ in previous publications, consult, e.g., the recent reviews [26] and [9], we present the following linear superiorization algorithm. The input to the algorithm consists of the problem data A, b, and c of (2) and (3), respectively, a user-chosen initialization point $\bar{y}$ and a kernel $0 < \alpha < 1$ (see item 1 in Subsection 3.3) with which the algorithm generates the step sizes $\beta_{k,n}$, as well as an integer N (see item 7 in Subsection 3.3). All quantities in the algorithm that have not yet been defined or explained are detailed in the Subsection 3.3 below.

Algorithm 4 ***Linear Superiorization (LinSup)***

set $k \leftarrow 0$

set $y^k \leftarrow \bar{y}$

set $\ell_{-1} \leftarrow 0$

while stopping rule not met **do**

  set $n \leftarrow 0$

  set $\ell \leftarrow \text{rand}(k, \ell_{k-1})$

  set $y^{k,n} \leftarrow y^k$

  while $n<N$ **do**

    set $\beta_{k,n} \leftarrow \eta_\ell$

    set $z \leftarrow y^{k,n} - \beta_{k,n} \frac{c}{\|c\|_2}$

    set $n \leftarrow n + 1$

    set $y^{k,n} \leftarrow z$

    set $\ell \leftarrow \ell + 1$

  end while

  set $\ell_k \leftarrow \ell$

  set $y^{k+1} \leftarrow \mathcal{A}\left(y^{k,N}\right)$

  set $k \leftarrow k + 1$

end while

3.2 The Agmon-Motzkin-Schoenberg algorithm as the basic algorithm

We use the projection method of Agmon-Motzkin-Schoenberg (AMS) [1, 32], see also, e.g., [19, Algorithm 5.4.2], as the basic algorithm for feasibility-seeking represented by $\mathcal{A}$ in step 16 of Algorithm 4. Denote the half-spaces represented by individual rows of (2) by H_i,

$$H_i := \{x \in R^J \mid \langle a^i, x\rangle \leq b_i\}, \tag{4}$$

where $a^i \in R^J$ is the i-th row of A and $b_i \in R$ is the i-th component of b in (2). The orthogonal projection of an arbitrary point $z \in R^J$ onto H_i, has the closed-form

$$P_{H_i}(z) = \begin{cases} z - \dfrac{\langle a^i, z\rangle - b_i}{\|a^i\|^2} a^i, & \text{if } \langle a^i, z\rangle > b_i, \\ z, & \text{if } \langle a^i, z\rangle \leq b_i. \end{cases} \tag{5}$$

Algorithm 5 ***The Relaxation Method of Agmon, Motzkin and Schoenberg (AMS)***.

Initialization: *$x^0 \in R^n$ is arbitrary.*

Iterative step: *Given the current iteration vector x^k the next iterate is calculated by*

$$x^{k+1} = \begin{cases} x^k - \lambda_k \dfrac{\langle a^{i(k)}, x^k\rangle - b_i}{\|a^{i(k)}\|^2} a^i, & \text{if } \langle a^{i(k)}, x^k\rangle > b_{i(k)}, \\ x^k, & \text{if } \langle a^{i(k)}, x^k\rangle \leq b_{i(k)}. \end{cases} \tag{6}$$

Relaxation parameters: *The parameters λ_k are such that $\epsilon_1 \leq \lambda_k \leq 2-\epsilon_2$, for all $k \geq 0$, with some, arbitrarily small, $\epsilon_1, \epsilon_2 > 0$.*

Control: *The control sequence $\{i(k)\}_{k=0}^{\infty}$ is almost cyclic on $\{1, 2, ..., I\}$.*

This AMS cyclic feasibility-seeking algorithm goes cyclically through the inequalities of (2). To handle the nonnegativity constraints in (2) we just take the current iteration vector in hand, after having done a full sweep of AMS through all I row-inequalities, and set its negative components to zero while keeping the others unchanged.

A corner stone of the superiorization methodology, in general as well as for the linear case discussed here, is the perturbation resilience of the basic algorithm that is used. The AMS algorithm is known to be bounded perturbation resilience, this can be obtained from various previously published results, see, e.g., [16, Theorem 12], [33].

3.3 Implementation details and explanations

Here are the implementation details of our experimental work with the LinSup Algorithm 4 presented in the next sections.

1. **Step-sizes of the perturbations.** The step sizes $\beta_{k,n}$ in Algorithm 4 must be such that $0 < \beta_{k,n} \leq 1$ in a way that guarantees that they form a summable sequence $\sum_{k=0}^{\infty} \sum_{n=0}^{N-1} \beta_{k,n} < \infty$, see, e.g., [18]. To this end Algorithm 4 assumes that we have available a summable sequence $\{\eta_\ell\}_{\ell=0}^{\infty}$ of positive real numbers generated by $\eta_\ell = \alpha^\ell$, where $0 < \alpha < 1$. Simultaneously with generating the iterative sequence $\{y^k\}_{k=0}^{\infty}$, a subsequence of $\{\eta_\ell\}_{\ell=0}^{\infty}$ is used to generate the step sizes $\beta_{k,n}$ in step 9 of Algorithm 4. The number α is called the kernel of the sequence $\{\eta_\ell\}_{\ell=0}^{\infty}$.

2. **Controlling the decrease of the step-sizes of target function reduction**. If during the application of Algorithm 4 the step sizes $\beta_{k,n}$ decrease too fast then too little leverage is allocated to the target function reduction activity that is interlaced into the feasibility-seeking activity of the basic algorithm. This delicate balance can be controlled by the choice of the index ℓ updates and separately by the value of α whose powers α^ℓ determine the step sizes $\beta_{k,n}$ in step 9. In our work we adopt a strategy for updating the index ℓ that was proposed and implemented for total variation (TV) image reconstruction from projections by Prommegger and by Langthaler in [38, page 38 and Table 7.1 on page 49] and in [29], respectively. Instead of consecutively increasing ℓ by taking its value as it was at the end of the last sweep of N perturbations and starting the new sweep from that last value, the Prommegger and Langthaler strategy advocates to set ℓ at the beginning of every new iteration sweep (steps 5 and 6) to a random number between the current iteration index k and the value of ℓ from the last iteration sweep, i.e., $\ell_k = \text{rand}(k, \ell_{k-1})$. This strategy was denoted in those reports by the name "ATL2" and having verified its utility for our work we adopted it in all our experiments. On the other hand, the value of α with whose powers α^ℓ the step sizes $\beta_{k,n}$ are determined was experimented with and our computational results in the sections below report on this for the experiments with LinSup that we performed. Obviously, there are no such delicate balances in the Simplex algorithm, although there were at the early stages (pivot

strategies, candidate list, etc.). Further work is needed to make LinSup more resistant to the choice of parameters.

3. **No target function value comparisons**. Influenced by the results of [18] we completely deleted from the algorithm the decision-making test that compares the target function value at z of step 10 in the perturbation inner-loop with the target function value at y^k. This decision-making test appeared in many previous formulations of the superiorized version of the basic algorithm, see, e.g., step (xiv) of the "Superiorized Version of Algorithm **P**" in [27, page 5537]. Since we were able to prove our experimental claims without this test and since the mathematical treatment in [18] also proceeded well without it we left it out. See also the Appendix in Section 7 below.

4. **The proximity function**. To measure the feasibility-violation (or level of agreement) of a point with respect to the target set M we used the following proximity function

$$\Pr(x) := \frac{1}{2I}\sum_{i=1}^{I}\frac{\left(\left(\langle a^i, x\rangle - b_i\right)_+\right)^2}{\sum_{j=1}^{J}\left(a_j^i\right)^2} + \frac{1}{2J}\sum_{j=1}^{J}\left(\left(-x_j\right)_+\right)^2 \tag{7}$$

   where the plus notation means, for any real number d, that $d_+ := \max(d, 0)$. The proximity function is scaling invariant because it measures exactly the weighted (with equal weights) sum of half square distances of the point x from all linear inequality constraints. These distances (see (5)) are geometric entities insensitive to scaling.

5. **Initialization points of the algorithms**. In our experimental studies the initialization point $\bar{y}$ was always chosen to be a non-feasible point of the target set, $\bar{y} \notin M$. Otherwise, there was the danger that the algorithm would not move because of the AMS feasibility-seeking step 16 of the algorithm. This was done in the following manner. First $\bar{y}$ is randomly picked in the interval $[0, 1]$ by the algorithm or chosen otherwise by the user and its proximity function (7) to the set M is calculated. If the proximity of $\bar{y}$ is zero then we redefine a new $\bar{y} \leftarrow 10\cdot\bar{y}$ and repeat these 10-fold increments until we found a point with nonzero proximity to serve as the initialization point.

6. **Test problem generation**. We created target sets M of various sizes and a linear target function ϕ and run on them Algorithm 4 with or without superiorization, as the case maybe, in our experimental work reported below. Each problem of given $I \times J$ size was created by defining a matrix A whose elements are randomly chosen within the interval $[-1, 2]$ for all experiments. A vector c was chosen randomly in $[-2, 3]$ always. To guarantee feasibility (nonemptiness) of the target set M, we defined b by $b := A\mathbf{1} + 10 \cdot \mathbf{1}$, where $\mathbf{1}$ is the vector of all ones, and this guarantees that $\mathbf{1} \in M$. In all experiments we used problems with the following $I \times J$ sizes: 80×100, 200×250, 400×500, $800 \times 1,000$, $2,000 \times 2,500$, $4,000 \times 5,000$, and $8,000 \times 10,000$.

   We have not controlled the sparsity of the test problems and have not investigated this issue. All we can say is that since all entries of the matrices were uniformly distributed in the interval [-1,2], the probability of any entry being equal to zero exactly is almost nonexistent. There are millions of entries in total and many of them are probably close to zero but overall it is accurate to say that the matrices that we generated were dense.

7. **The number N of perturbation steps**. This number N of perturbation steps that are performed prior to each application of the feasibility-seeking operator $\mathcal{A}$ (in step 16) affects the performance of the LinSup algorithm. It influences the balance between the amounts of computations allocated to feasibility-seeking and those allocated to target function reduction steps. A too large N will make Algorithm 4 spend too much resources on the perturbations that yield target function reduction. In order to find an appropriate value of N for our work we created 10 problems of each of the problem sizes 80×100, 200×250, 400×500, $800 \times 1,000$, $2,000 \times 2,500$, and 3 problems of the problem size $4,000 \times 5,000$. We applied Algorithm 4 to each problem with the number N being allowed to vary in the range from $N = 5$ to $N = 100$. All other parameters except N were kept equal in all runs (specifically, with kernel $\alpha = 0.99$, relaxation parameters in the AMS algorithm in step 16 $\lambda_k = \lambda = 1$ for all $k \geq 0$, and initialization point $\bar{y} = 10 \cdot \mathbf{1}$ of appropriate size for all problems.) The stopping rule for these experiments was when the proximity function $\Pr(x)$ of (7) dropped below the value $\varepsilon = 10^{-10}$. We recorded for all runs the relative errors between the linear target function value ϕ_{LinSup} obtained by LinSup when it

was stopped, and the linear objective function value ϕ_{Simplex} obtained by the Simplex method when MATLAB reported the solution has been reached, given by

$$RE := \frac{|\phi_{\text{LinSup}} - \phi_{\text{Simplex}}|}{|\phi_{\text{Simplex}}|}. \tag{8}$$

The table in Figure 1 contains averaged values of those relative errors averaged over all problems of the same size. These data are plotted in Figure 2. Based on these findings we decided to use $N = 30$ in all our subsequent computational experiments due to the observation that, for all problem sizes, the decrease of relative error RE became small beyond this value of N.

8. **The relaxation parameters in the AMS algorithm**. In our experiments we set all relaxation parameters in the AMS feasibility-seeking algorithm represented by the algorithmic operator $\mathcal{A}$ and embodied in step 16 of Algorithm 4 to $\lambda_k = \lambda = 1$. When doing only feasibility-seeking proper with the AMS algorithm it has been frequently shown in the literature that the relaxation parameters have significant effect on the behavior of the algorithm, see, e.g., [25, Subsections 11.2 and 11.5]. However, here when the AMS algorithm is embedded in the LinSup algorithm we observed that the relaxation parameters in the AMS algorithm have a weak influence on the overall behavior and, therefore, they were set, at this phase of the work, to 1.

9. **Handling the nonnegativity constraints**. As mentioned above, the nonnegativity constraints in (2) are handled by taking the current iteration vector in hand after having done a full sweep of AMS through all I row-inequalities of (2) and setting its negative components to zero while keeping the others unchanged.

4 Experimental Task 1: Linear superiorization finds a superior feasible point

Any of the large variety of projection methods to handle linear inequality constraints feasibility-seeking can be used, but we choose for the basic algorithm $\mathcal{A}$ the famous Agmon-Motzkin-Schoenberg (AMS) cyclic feasibility-seeking projection method [1, 32], known in the image reconstruction literature as Algebraic Reconstruction Technique (ART) for inequalities [25, Subsection 11.2], see also [19, Algorithm 5.4.2].

Our aim in Task 1 is to experimentally validate or reject the following claim:

Claim 6 *Consider two runs of the LinSup Algorithm 4 for the same target set M as in (2), one with and the other without superiorization. "Without superiorization" means that steps 5–15 in Algorithm 4 are deleted and in step 16 one takes $y^{k,N} = y^k$ which amounts to only running the feasibility-seeking basic algorithm $\mathcal{A}$ without any perturbations. Assume that other than that everything else is equal in the two runs, such as the initialization point $\bar{y}$ and all parameters associated with the application of the feasibility-seeking basic algorithm $\mathcal{A}$ in step 16, as well as the stopping rule. Under these circumstances the run "with superiorization" will yield (i.e., stop at) a point y^* whose $\phi(y^*) := \langle c, y^* \rangle$ value will be smaller than $\phi(y^{**})$ of a point y^{**} at which the run "without superiorization" would stop.*

To prove this claim we created test problems as described in item 6 in Subsection 3.3. On each such problem we ran LinSup without superiorization and with superiorization and discovered that in all our experiments Claim 6 is true. We ran all experiments with kernel $\alpha = 0.99$, relaxation parameters in the AMS algorithm in step 16 $\lambda_k = \lambda = 1$ for all $k \geq 0$, and initialization point $\bar{y} = 10 \cdot \mathbf{1}$ of appropriate size for all problems. The stopping rule for these experiments was when the proximity function $\Pr(x)$ of (7) dropped below the value $\varepsilon = 10^{-20}$. The number N of perturbation steps that are performed prior to each application of the feasibility-seeking operator $\mathcal{A}$ (in step 16) was, as decided in item 7 in Subsection 3.3, $N = 30$. The execution times in seconds, shown in the table in Figure 3, naturally show that superiorization needs more time than plain feasibility-seeking. All values in this table are averaged over 10 different problems for each problem size except for the last one $(8,000 \times 10,000)$ for which we made only one run. The right-hand side columns in the table show the truth of our Claim 6. These data are plotted in Figure 4 and one can clearly note that the trend persists and strengthens as the problem sizes increase.

Having generated our data as described in item 6 of Subsection 3.3, the target function value actually depends also on the size J of the vector x. It is observed from the table in Figure 3 that when this size increases 10 times, the corresponding target function values with and without superiorization both roughly increase 10 times as well. From this point of view, the relative gap between the target function values with and without superiorization is consistent for different problem sizes.

5 Experimental Task 2: Linear superiorization versus linear optimization with the Simplex algorithm

To compare the performance of LinSup with that of a linear optimization algorithm we used MATLAB [31] and chose the 'Simplex' algorithm from the 'linprog' solver. We created test problems as described in item 6 in Subsection 3.3. Since we wish to compare with the outputs and execution times of the Simplex algorithm we first let MATLAB's Simplex algorithm run on each test problem to ascertain that the 'exitflag' that it yields is 'Function converged to a solution x'. If the test problem turned out to be not solvable by the Simplex algorithm we discarded it in favor of another test problem generated as described in item 6 in Subsection 3.3 for which Simplex outputs a solution.

Once a test problem was solved by Simplex we calculated the proximity $\Pr(x)$ of (7) of the solution provided by the Simplex algorithm, which was generally small. This proximity value was then used as the stopping rule for the LinSup run on the same problem. When LinSup reached this proximity it stopped and the iterate at stopping was its output solution.

Having forced the LinSup to run until it reached the same proximity as the solution obtained by the Simplex algorithm, we recorded and compared the target function values and the execution times for both. Based on the experience gained in numerous experiments and runs we made decisions that fixed all parameters except for one and report here on the performance of LinSup and MATLAB's Simplex algorithm for several values of this parameter and for different problem sizes. As said above, the number N of perturbation steps that are performed prior to each application of the feasibility-seeking operator $\mathcal{A}$ was fixed to $N = 30$ in all experiments. The feasibility-seeking operator $\mathcal{A}$ (in step 16) was the AMS algorithm of Subsection 3.2 with fixed relaxation parameters $\lambda_k = \lambda = 1$ as in item 8 in Subsection 3.3. All other implementation details were as in Subsection 3.3. We explored the effect of different choices of the kernel α (in item 1 in Subsection 3.3) on all runs.

All data presented in the following tables and plots is averaged over 10 different and independently-generated problems for each size from 80×100 to $2,000 \times 2,500$, 5 different and independently-generated problems of size $4,000 \times 5,000$ and one problem of size $8,000 \times 10,000$.

In addition to the relative error RE of (8) we recorded here also the time ratio

$$TR := \frac{\text{execution time of LinSup}}{\text{execution time of Simplex}}. \tag{9}$$

5.1 The results and what they tell us

The table in Figure 5 shows target function $\phi(x) = \langle c, x \rangle$ values for the Simplex algorithm alongside with the target function values outputs by LinSup at stopping for 3 different values of the kernel α. The relative errors RE of (8) are also shown. With larger values of the kernel α its powers diminish slower, leaving more room for the target function reduction perturbations to affect the outcome of LinSup. As the problem sizes increase however the relative error RE also increases.

The table in Figure 6 shows execution times in seconds for the Simplex algorithm alongside with those of LinSup for 3 different values of the kernel α. The time ratios TR of (9) are also shown. Here one observes that LinSup is fast compared to the time of the Simplex algorithm.

The Figures 7–11 are based on the data in the tables of Figures 5–6. Plots of relative errors RE versus problem sizes for LinSup with 3 different kernel α values based on the data from the table in Figure 5 appear in Figure 7. For each α the relative error increases with the increase of problem sizes. For all problem sizes the relative errors decrease with increasing value of α. For all problem sizes the relative error is smaller for the larger value of $\alpha = 0.999$.

Plots of time ratios TR versus problem sizes for LinSup with 3 different kernel α values based on the data from the table in Figure 6 appear in Figure 8. For each α the time ratio decreases with the increase of problem sizes. For all problem sizes the time ratios decrease for decreasing value of α. This draws our attention to the emerging conflict of choosing the kernel α. For better (smaller) relative error choose it larger but for better (smaller) time ratio choose it smaller. We see these trade-offs in the next figures as well.

Figure 9 shows target function values plotted against problem sizes for the 3 values of the kernel α. The larger $\alpha = 0.999$ allows for more resource investment of the LinSup algorithm into function reduction steps. Thus, it yields target function values that are closer to those obtained from the Simplex algorithm. Figure 10 tells the story in a nutshell by superimposing Figures 7 and 8. This shows graphically the trade-off between target function value reduction and speed in the LinSup algorithm.

Execution times in thousands of seconds versus problem sizes of the Simplex algorithm and of LinSup for 3 kernel α values are depicted in Figure 11. Observe the steep increase in time of the Simplex algorithm (dashed line) for the larger sized problem. LinSup is more moderate in the growth of needed execution times vis-a-vis the Simplex algorithm.

Our results show that there is a built-in "conflict" in choosing the parameters that govern the delicate balance between the efforts that the LinSup algorithm invests in feasibility-seeking and in function reduction with perturbations. But the behavior of these results along increasing problem sizes leave room to hope that with further increase of problem sizes LinSup will gain more ground and become even a competitor to linear minimization algorithms. Observe that for the problem in the last row of the tables in Figures 5 and 6 LinSup with $\alpha = 0.999$ stops at target function value quite close to the one obtained by the Simplex algorithm at about one third of the time it took the Simplex algorithm.

5.2 Allowing the Simplex to terminate suboptimally

LinSup is not intended to solve the LP problem but, as explained in Section 2, to provide a feasible point with reduced (not necessarily minimal) linear target function value. However, from the point of view of the LP problem an output of LinSup can be considered a "reasonably good approximate solution of the LP problem". This raises the question, suggested by a referee, how would this compare with a suboptimally terminated Simplex run. To take a preliminary look at this issue we generated a $8,000 \times 10,000$ LP problem and let Simplex and LinSup run on it. The LinSup was stopped when its iterates showed no further significant changes (i.e., when $\frac{\left\|x^{k+1} - x^k\right\|}{\left\|x^k\right\|} \leq 10^{-16}$) and it was run with two different values of $\alpha = 0.99$ and $\alpha = 0.995$. The Simplex was not allowed to run until optimality but stopped at a time that is just a little longer than the time it took the LinSup runs to stop. These stopping decisions enable us to compare LinSup with a suboptimally stopped Simplex on this problem. Proximity function and linear target function calculation times after each iteration were subtracted from the Simplex run times because they are not an integral part of Simplex.

These results are depicted in Figures 12 and 13. Although far from being fully explored, the results show that if this Simplex run would have been stopped suboptimally, say after 5,000 seconds, both runs of the LinSup would have yielded lower linear target function values, as seen in Figure 12. At this point in time Simplex would have delivered an output with better feasibility, i.e., lower proximity value. However, at a later point in time, say after 20,000 seconds, both LinSup runs would have a lower proximity than the Simplex as seen in Figure 13 and the one with the higher kernel value α would even have a lower linear target function value.

This hints at the possible advantages of LinSup for large LP problems. Looking at the output of LinSup as a "reasonably good approximate solution of the LP problem", LinSup not only converges to such a solution faster than it takes Simplex to solve a problem to machine precision accuracy, but also faster than a suboptimally stopped Simplex. Admittedly, this and the other experiments presented here call for further work, see Section 6.

6 Conclusions

Linear superiorization (LinSup) is not, as far as we know at this time, a minimization method. Finding a constrained minimum point with it cannot be guaranteed. What it does is to steer feasibility-seeking algorithms toward points with lesser (not necessarily minimal) linear target function values. The computationally-efficient feasibility-seeking algorithms that use projections onto the convex closed sets of the constraints, embodied in LinSup, are particularly successful for the linear case. The perturbations to reduce the linear target function values need no effort other than using $-c$ as a direction of descent. Therefore, previous work on the superiorization methodology in general (see the references mentioned in the Introduction and in the Appendix) along with the proof of concept experimental work presented here suggest that LinSup is potentially a viable option to handle large LP problems.

Our results show that LinSup indeed finds a superior feasible point. That the increase in execution times as function of problem sizes of LinSup is more moderate than that of the Simplex algorithm. This motivates us in formulating the following conjecture.

Conjecture 7 *There exists a size-level of huge problems above which LinSup will perform better than linear minimization algorithms. Maybe that this will call for using feasibility-seeking projection methods inside LinSup that lend themselves to parallelization, such as block-iterative projections (BIP) or string-averaging projections (SAP) methods mentioned in Subsection 3.1 above.*

Many questions present themselves for further research based on the current work. Here is a telegraphic list of some potentially interesting directions:

(i) Expand the computational work to larger problem sizes and differently generated problems.

(ii) Test LinSup on a larger class of test problems than those used here such as LP benchmark test problems from Netlib (http://www.netlib.org/) or other repositories.

(iii) Study LinSup with additional feasibility-seeking projection methods that lend themselves to parallelization, such as block-iterative projections (BIP) or string-averaging projections (SAP) methods.

(iv) Investigate the parameters' effects on the behavior of LinSup by repeating experiments with different values of: The number N of perturbation steps that are performed prior to each application of the feasibility-seeking operator $\mathcal{A}$, the relaxation parameters λ_k in the feasibility-seeking embedded basic algorithm, the kernel α with which the step-sizes $\beta_{k,n}$ are generated.

(v) Advance the mathematical analysis of LinSup.

(vi) Repeat the above comparisons for additional linear optimization algorithms such as 'interior-point' or 'active-set' in MATLAB or others.

(vii) Investigate the inconsistent case wherein the target set M of (2) is empty and is replaced, e.g., by the set of closest points to all constraints according to some proximity function. Linear programming algorithms might not work but LinSup can still furnish a useful result.

(viii) Study LinSup for sparse linear constraints for which some projection methods have already demonstrated their effectiveness as feasibility-seeking algorithms.

7 Appendix: The algorithmic evolution of superiorization

The algorithmic structure of the superiorized version of a basic algorithm has undergone changes and modifications over the past several years since it inception. All changes preserve the underlying basic methodology and it is useful to briefly review them here. In [5] superiorization appeared although the words superiorization and perturbation resilience were not yet in use there. It built on some earlier theoretical work in [6, 7]. The pseudocode on the right-hand side column of page 543 in [5] constitutes the first superiorization algorithm. The step sizes β there (line 9) are simply halved and there is one function reduction step (line 3) for each sweep of the feasibility-seeking algorithm $\mathbf{P}$ (in line 6). There are two decision-making steps (on lines 4 and 7). This algorithm was used in Scott Penfold's thesis work [35] (see also [36]) and in the paper [37]. The Algorithms 2 and 3 (TVS1-DROP and TVS2-DROP, respectively) in [37] relate to different variants of the built-in feasibility-seeking algorithm DROP of [14], not to different superiorization methods. In the same paper [37] the expensive decision-making step (line 12) in both algorithms was removed without adverse effects thus allowing significant time savings.

In the "Superiorized version of algorithm $\mathbf{P}$" on page 6 of [13] both decision-making steps from earlier versions still appear, this time in a single line (line xiii). However, in the subsequent [27] the expensive decision-making step of line 12 in [37] is not to be seen anymore and, additionally, the negative subgradient (negative gradient if the function is differentiable) is replaced by any direction of "non-ascend" for the function ϕ that is superiorized. This last variation is not needed for total variation (TV) superiorization but might come to use for other functions ϕ. Another new ingredient in [27] which is very useful is the ability to do in an inner loop (from line vii till line xvii) N steps of function reduction (N is user-determined) for each sweep of the feasibility-seeking algorithm, denoted by $\mathbf{P}_T$ there (in line xviii).

An interesting comparative study appears in [15]. The algorithm there is called "Superiorized Version of the Basic Algorithm" and appears on pp. 737–738 wherein the feasibility-seeking algorithm is denoted by A_C (in step 18).

In [18, Algorithm 4.1] another step forward was made by (i) allowing the number N (of [27]) to vary from one iteration to another so that N is replaced by N_k where k is the iteration index, and (ii) discarding the second decision-making check that was on line 14 in [15] and in earlier algorithms for superiorization. The replacement of N by N_k is mathematically valid but, to the best of our knowledge, has not yet been experimented with by anyone.

Two important recent works on implementations of the superiorization algorithm appear in [29] and [38]. One important additional modification in those, that we adopted in our work (see item 2 in Subsection 3.3 above), is the way of controlling the perturbations' step-sizes $\beta_{k,n}$ in Algorithm 4 via a special strategy of updating the index ℓ at each sweep of iterations.

Acknowledgments. We gratefully acknowledge some preliminary discussions with Ran Davidi and John Chinneck. We thank the two anonymous referees for their constructive comments which helped us improve the paper. The MATLAB programming was skillfully performed with great enthusiasm and devotion by Yehuda Zur, for which we are indebted to him. This work was supported by Research Grant No. 2013003 of the United States-Israel Binational Science Foundation (BSF) and by Award No. 1P20183640-01A1 of the National Cancer Institute (NCI) of the National Institutes of Health (NIH).

Comment. Final version preprints of the author's papers cited in the references list below are available at: http://math.haifa.ac.il/yair/censor-recent-pubs.html. Other papers on superiorization cited below have their abstracts and DOI codes posted on: http://math.haifa.ac.il/yair/bib-superiorization-censor.html.

References

[1] S. Agmon, The relaxation method for linear inequalities, *Canadian Journal of Mathematics* **6** (1954), 382–392.

[2] J.-B. Baillon, P.L. Combettes and R. Cominetti, There is no variational characterization of the cycles in the method of periodic projections, *Journal of Functional Analysis* **262** (2012), 400–408.

[3] H.H. Bauschke and J.M. Borwein, On projection algorithms for solving convex feasibility problems, *SIAM Review* **38** (1996), 367–426.

[4] H.H. Bauschke and V.R. Koch, Projection methods: Swiss army knives for solving feasibility and best approximation problems with half-spaces, *Contemporary Mathematics* **636** (2015), 1–40.

[5] D. Butnariu, R. Davidi, G.T. Herman, and I.G. Kazantsev, Stable convergence behavior under summable perturbations of a class of projection methods for convex feasibility and optimization problems, *IEEE Journal of Selected Topics in Signal Processing* **1** (2007), 540–547.

[6] D. Butnariu, S. Reich and A.J. Zaslavski, Convergence to fixed points of inexact orbits of Bregman-monotone and of nonexpansive operators in Banach spaces, in: H.F. Nathansky, B.G. de Buen, K. Goebel, W.A. Kirk, and B. Sims, *Fixed Point Theory and its Applications*, (Conference Proceedings, Guanajuato, Mexico, 2005), Yokahama Publishers, Yokahama, Japan, pp. 11–32, 2006.

[7] D. Butnariu, S. Reich and A.J. Zaslavski, Stable convergence theorems for infinite products and powers of nonexpansive mappings, *Numerical Functional Analysis and Optimization* **29** (2008), 304–323.

[8] A. Cegielski, *Iterative Methods for Fixed Point Problems in Hilbert Spaces*, Lecture Notes in mathematics 2057, Springer-Verlag, Berlin, Heidelberg, Germany, 2012.

[9] Y. Censor, Weak and strong superiorization: Between feasibility-seeking and minimization, *Analele Stiintifice ale Universitatii Ovidius Constanta-Seria Matematica* **23** (2015), 41–54.

[10] Y. Censor, "Superiorization and Perturbation Resilience of Algorithms: A Bibliography compiled and continuously updated by Yair Censor", an Internet page: http://math.haifa.ac.il/yair/bib-superiorization-censor.html.

[11] Y. Censor and A. Cegielski, Projection methods: an annotated bibliography of books and reviews, *Optimization* **64** (2015), 2343–2358. DOI:10.1080/02331934.2014.957701.

[12] Y. Censor, W. Chen, P.L. Combettes, R. Davidi and G.T. Herman, On the effectiveness of projection methods for convex feasibility problems with linear inequality constraints, *Computational Optimization and Applications* **51** (2012), 1065–1088.

[13] Y. Censor, R. Davidi and G.T. Herman, Perturbation resilience and superiorization of iterative algorithms, *Inverse Problems* **26** (2010), 065008 (12pp).

[14] Y. Censor, T. Elfving, G.T. Herman and T. Nikazad, On diagonally-relaxed orthogonal projection methods, *SIAM Journal on Scientific Computing* **30** (2008), 473–504.

[15] Y. Censor, R. Davidi, G.T. Herman, R.W. Schulte and L. Tetruashvili, Projected subgradient minimization versus superiorization, *Journal of Optimization Theory and Applications* **160** (2014), 730–747.

[16] Y. Censor and A.J. Zaslavski, Convergence and perturbation resilience of dynamic string-averaging projection methods, *Computational Optimization and Applications* **54** (2013), 65–76.

[17] Y. Censor and A.J. Zaslavski, String-averaging projected subgradient methods for constrained minimization, *Optimization Methods & Software* **29** (2014), 658–670.

[18] Y. Censor and A.J. Zaslavski, Strict Fejér monotonicity by superiorization of feasibility-seeking projection methods, *Journal of Optimization Theory and Applications* **165** (2015), 172–187.

[19] Y. Censor and S.A. Zenios, *Parallel Optimization: Theory, Algorithms, and Applications*, Oxford University Press, New York, NY, USA, 1997.

[20] J.W. Chinneck, *Feasibility and Infeasibility in Optimization*, Springer Science+Business Media, LLC, New York, NY, USA, 2008.

[21] P.L. Combettes, On the numerical robustness of the parallel projection method in signal synthesis, *IEEE Signal Processing Letters* **8** (2001), 45–47.

[22] R. Davidi, G.T. Herman, and Y. Censor, Perturbation-resilient block-iterative projection methods with application to image reconstruction from projections, *International Transactions in Operational Research* **16** (2009), 505–524.

[23] E. Garduño, and G.T. Herman, Superiorization of the ML-EM algorithm, *IEEE Transactions on Nuclear Science* **61** (2014), 162–172.

[24] D. Gordon and R. Gordon, Component-averaged row projections: A robust, block-parallel scheme for sparse linear systems, *SIAM Journal on Scientific Computing* **27** (2005), 1092–1117.

[25] G.T. Herman, *Fundamentals of Computerized Tomography: Image Reconstruction from Projections*, Springer-Verlag, London, UK, 2nd Edition, 2009.

[26] G.T. Herman, Superiorization for image analysis, in: *Combinatorial Image Analysis*, Lecture Notes in Computer Science Vol. **8466**, Springer, 2014, pp. 1–7.

[27] G.T. Herman, E. Garduño, R. Davidi and Y. Censor, Superiorization: An optimization heuristic for medical physics, *Medical Physics* **39** (2012), 5532–5546.

[28] W. Jin, Y. Censor and M. Jiang, A heuristic superiorization-like approach to bioluminescence, *International Federation for Medical and Biological Engineering (IFMBE) Proceedings* **39** (2013), 1026–1029.

[29] O. Langthaler, *Incorporation of the Superiorization Methodology into Biomedical Imaging Software*, Marshall Plan Scholarship Report, Salzburg University of Applied Sciences, Salzburg, Austria, and The Graduate Center of the City University of New York, NY, USA, September 2014, (76 pages).
http://www.marshallplan.at/images/papers_scholarship/2014/Salzburg_University_of_Applied_Sciences_LangthalerOliver_2014.pdf.

[30] S. Luo and T. Zhou, Superiorization of EM algorithm and its application in single-photon emission computed tomography (SPECT), *Inverse Problems and Imaging* **8** (2014), 223–246.

[31] MATLAB®, A high-level language and interactive environment system by MathWorks, http://www.mathworks.com/products/matlab/.

[32] T.S. Motzkin and I.J. Schoenberg, The relaxation method for linear inequalities, *Canadian Journal of Mathematics* **6** (1954), 393–404.

[33] T. Nikazad, R. Davidi and G.T. Herman, Accelerated perturbation-resilient block-iterative projection methods with application to image reconstruction, *Inverse Problems* **28** (2012), 035005 (19pp).

[34] E.A. Nurminski, Single-projection procedure for linear optimization, *Journal of Global Optimization*, accepted for publication (2015). DOI: 10.1007/s10898-015-0337-9.

[35] S.N. Penfold, *Image Reconstruction and Monte Carlo Simulations in the Development of Proton Computed Tomography for Applications in Proton Radiation Therapy*, PhD Thesis, University of Wollongong, Wollongong NSW 2522, Australia, 2010.

[36] S.N. Penfold, *A Prototype Proton Computed Tomography System: Image Reconstruction and Monte Carlo Simulations*, Lambert Academic Publishing (LAP), Germany, 2012.

[37] S.N. Penfold, R.W. Schulte, Y. Censor and A.B. Rosenfeld, Total variation superiorization schemes in proton computed tomography image reconstruction, *Medical Physics* **37** (2010), 5887–5895.

[38] B. Prommegger, *Verification and Evaluation of Superiorized Algorithms Used in Biomedical Imaging: Comparison of Iterative Algorithms With and Without Superiorization for Image Reconstruction from Projections*, Marshall Plan Scholarship Report, Salzburg University of Applied Sciences, Salzburg, Austria, and The Graduate Center of the City University of New York, NY, USA, October 2014, (84 pages). http://www.marshallplan.at/images/papers_scholarship/2014/Salzburg_University_of_Applied_Sciences_PrommeggerBernhard_2014.pdf.

[39] M.J. Schrapp and G.T. Herman, Data fusion in X-ray computed tomography using a superiorization approach, *Review of Scientific Instruments* **85** (2014), 053701 (9pp).

[40] H.A. Simon, Rational choice and the structure of the environment, *Psychological Review* **63** (1956), 129–138.

Problem size	80X100	200X250	400X500	800X1,000	2,000X2,500	4,000X5,000
N=5	0.00585	0.01591	0.03671	0.07238	0.15490	0.24344
N=10	0.00416	0.00591	0.01460	0.03264	0.07839	0.12844
N=15	0.00484	0.00478	0.01071	0.02299	0.05430	0.08978
N=20	0.00524	0.00478	0.00983	0.01930	0.04439	0.07373
N=25	0.00545	0.00494	0.00979	0.01783	0.03964	0.06525
N=30	0.00569	0.00511	0.01009	0.01724	0.03641	0.05934
N=40	0.00611	0.00566	0.01063	0.01758	0.03387	0.05459
N=50	0.00626	0.00609	0.01135	0.01794	0.03342	0.05157
N=60	0.00637	0.00630	0.01173	0.01853	0.03373	0.05027
N=70	0.00649	0.00691	0.01237	0.01950	0.03345	0.04956
N=80	0.00653	0.00722	0.01267	0.02106	0.03380	0.04877
N=90	0.00665	0.00755	0.01297	0.02094	0.03450	0.04905
N=100	0.00668	0.00761	0.01336	0.02142	0.03551	0.05094

Figure 1: The realtive errors $|\phi_{\text{LinSup}} - \phi_{\text{Simplex}}| \,/\, |\phi_{\text{Simplex}}|$ between the linear target function value ϕ_{LinSup} obtained by LinSup when it was stopped and the linear objective function value ϕ_{Simplex} obtained by the Simplex method when MATLAB reported the solution has been reached, for different values of the number N of step 8 of Algorithm 4. The numbers in the table are average values over several problems of each size (see the text).

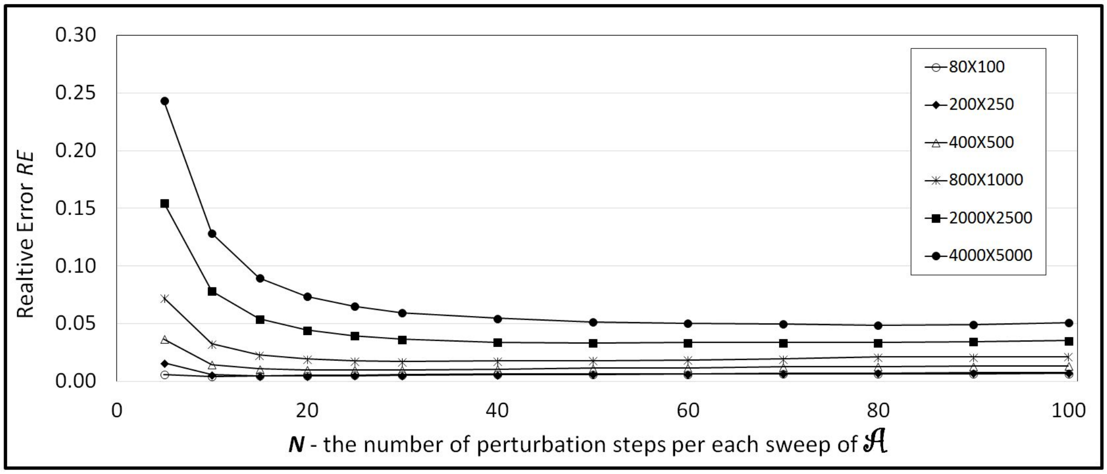

Figure 2: Plots of the data included in the table of Figure 1. Based on these findings we decided to use $N = 30$ in all our subsequent computational experiments.

Problem size	Time (seconds)		Target function	
	Superiorized	Unsuperiorized	Superiorized	Unsuperiorized
80X100	1.457	0.037	-136.598	48.630
200X250	3.777	0.106	-341.664	97.976
400X500	9.008	0.319	-697.631	163.639
800X1,000	49.454	1.516	-1433.237	438.540
2,000X2,500	358.088	11.192	-3755.759	1024.235
4,000X5,000	1778.331	54.924	-7619.119	2140.007
8,000X10,000	7979.140	330.171	-15145.999	4493.292

Figure 3: All values in this table are averaged over 10 different problems for each problem size except for the last one $(8,000 \times 10,000)$ for which we made only one run. The execution times in seconds naturally show that superiorization needs more time than plain feasibility-seeking. The two right-hand side columns in the table confirm the truth of our Claim 6 for the experiments that we performed.

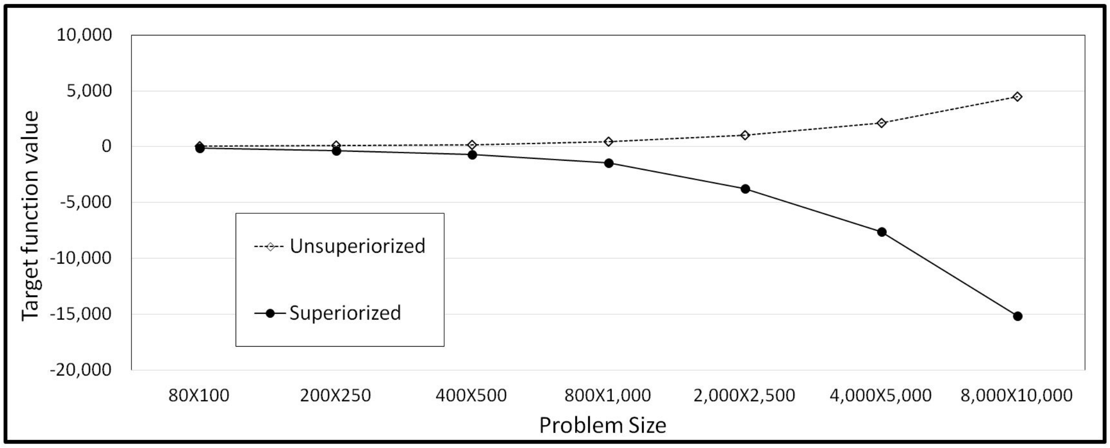

Figure 4: The data from the table in Figure 3 is plotted and shows that the gap between the target function values with and without superiorization steadily increases with the increase in problem sizes.

	Simplex	**LinSup**					
Problem size	**Target function**	**α=0.9**		**α=0.99**		**α=0.999**	
		Target function	**Relative error**	**Target function**	**Relative error**	**Target function**	**Relative error**
80X100	-133.516	-48.549	0.653	-132.995	0.004	-133.489	0.0002
200X250	-343.336	-76.725	0.775	-340.707	0.008	-343.170	0.0005
400X500	-708.006	-25.459	0.965	-700.540	0.011	-707.451	0.0008
800X1,000	-1469.552	99.394	1.068	-1442.833	0.018	-1467.610	0.0013
2,000X2,500	-3868.316	640.101	1.165	-3725.371	0.037	-3858.016	0.0027
4,000X5,000	-8110.310	1524.131	1.188	-7654.266	0.056	-8078.899	0.0039
8,000X10,000	-16779.901	2806.726	1.167	-15333.227	0.086	-16659.832	0.0072

Figure 5: This table shows the target function $\phi(x) = \langle c, x \rangle$ values for the Simplex algorithm alongside with the target function values outputs by LinSup at stopping for 3 different values of the kernel α. The relative errors RE of (8) are also shown.

	Simplex	LinSup					
Problem size	Time (seconds)	α=0.9		α=0.99		α=0.999	
		Time (seconds)	Time ratio	Time (seconds)	Time ratio	Time (seconds)	Time ratio
80X100	0.06	0.21	3.395	2.16	34.452	21.51	343.588
200X250	0.26	0.50	1.894	5.53	20.953	55.50	210.322
400X500	2.40	1.23	0.513	12.70	5.296	126.53	52.770
800X1,000	20.68	6.54	0.316	74.69	3.611	754.02	36.454
2,000X2,500	633.79	40.52	0.064	492.93	0.778	5111.70	8.065
4,000X5,000	11971.93	135.35	0.011	1813.56	0.151	19220.67	1.605
8,000X10,000	203756.69	709.36	0.003	7044.85	0.035	72083.44	0.354

Figure 6: This table shows execution times in seconds for the Simplex algorithm alongside with those of LinSup for 3 different values of the kernel α. The time ratios TR of (9) are also shown.

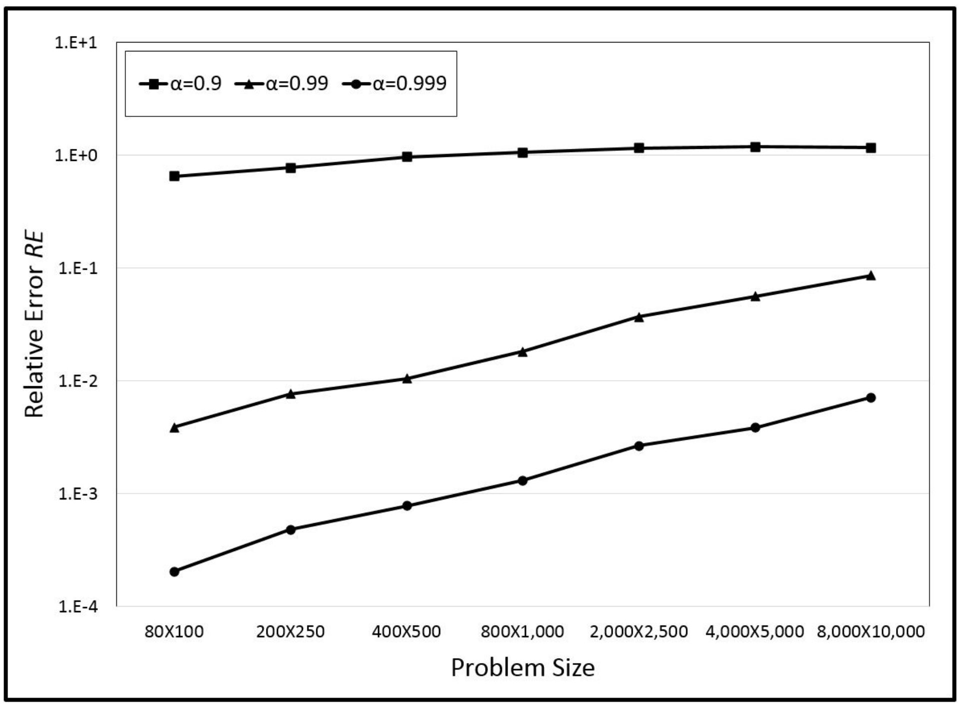

Figure 7: Plots of relative errors RE, on a logarithmic scale, versus problem sizes for LinSup with 3 different kernel α values based on the data from the table in Figure 5. For each α the relative error increases with the increase of problem sizes. For all problem sizes the relative errors decrease with increasing value of α.

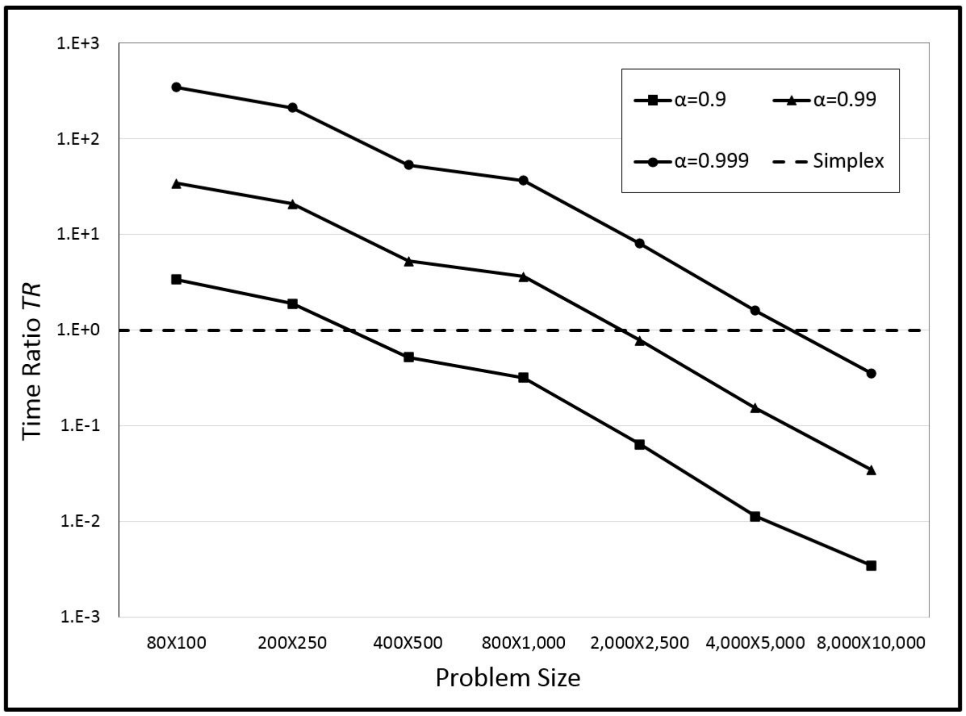

Figure 8: Plots of time ratios TR, on a logarithmic scale, versus problem sizes for LinSup with 3 different kernel α values based on the data from the table in Figure 6. For each α the time ratio dcreases with the increase of problem sizes. For all problem sizes the time ratios decrease for dcreasing value of α.

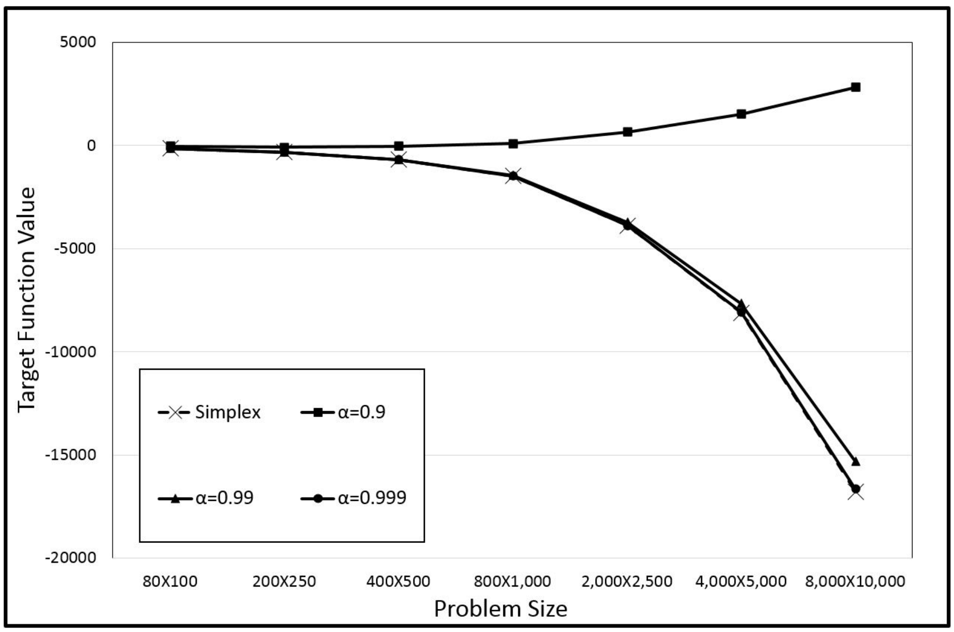

Figure 9: Target function values plotted against problem sizes for the 3 values of the kernel α. The larger $\alpha = 0.999$ allows more resource investment of the LinSup algorithm into function reduction steps. thus, yields target function values that are close to those obtained from the Simplex algorithm.

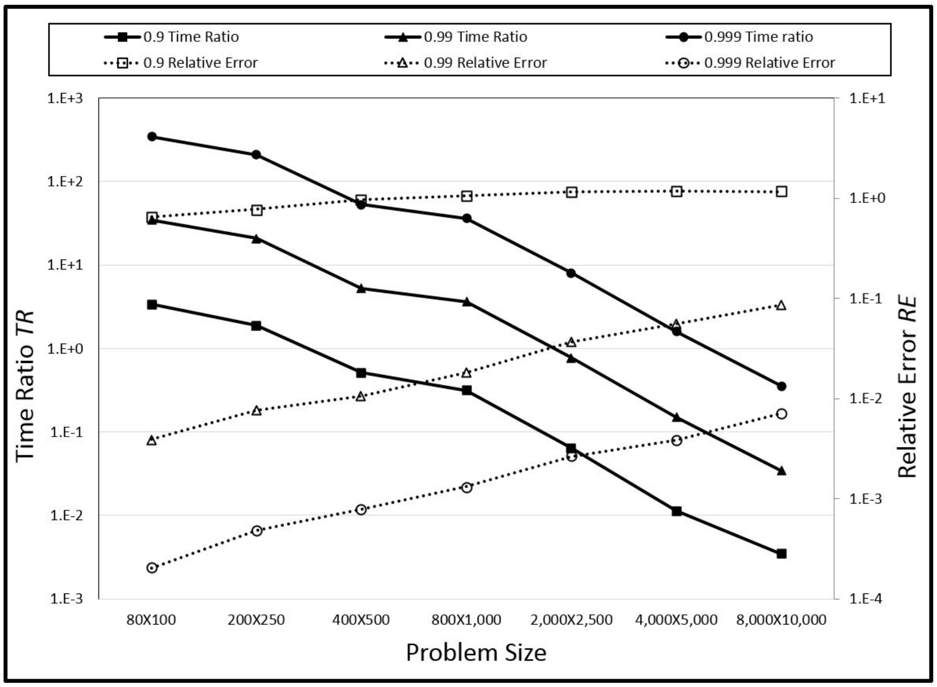

Figure 10: Here are the Figures 7 and 8 superimposed. This shows graphically the trade-off between target function value reduction and speed in the LinSup algorithm.

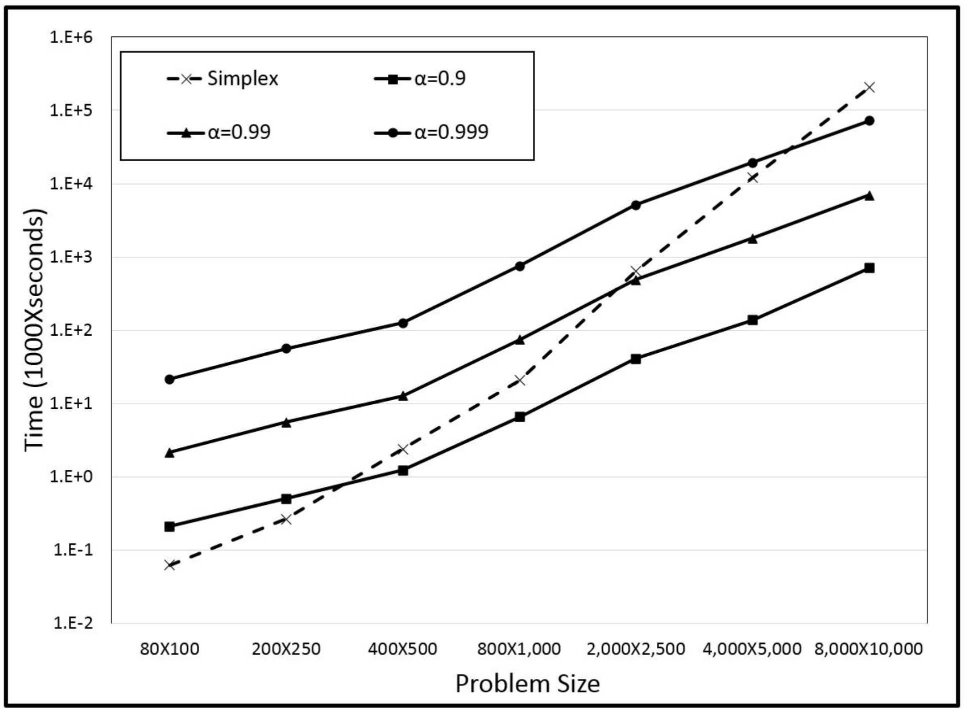

Figure 11: Execution times in thousands of seconds, on a logarithmic scale, versus problem sizes of the Simplex algorithm and of LinSup for 3 kernel α values. Observe the steep increase in time of the Simplex algorithm (dashed line).

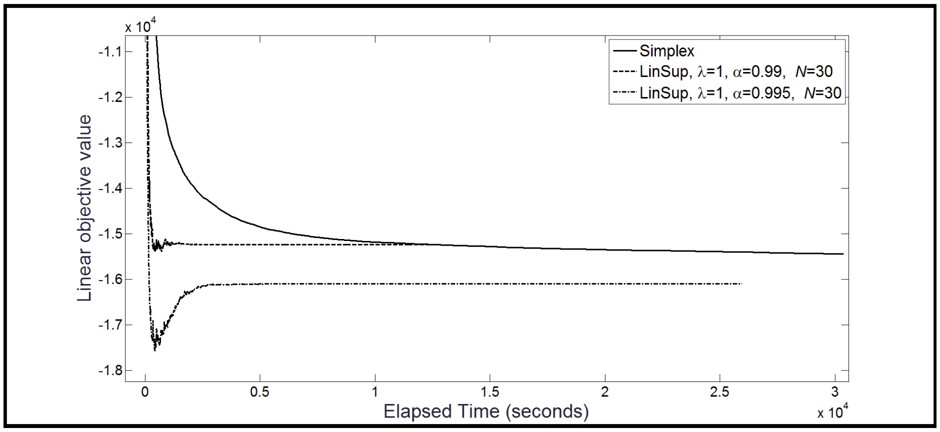

Figure 12: If this Simplex run would have been stopped suboptimally, say after 5,000 seconds, both runs of the LinSup would have yielded lower linear target function values.

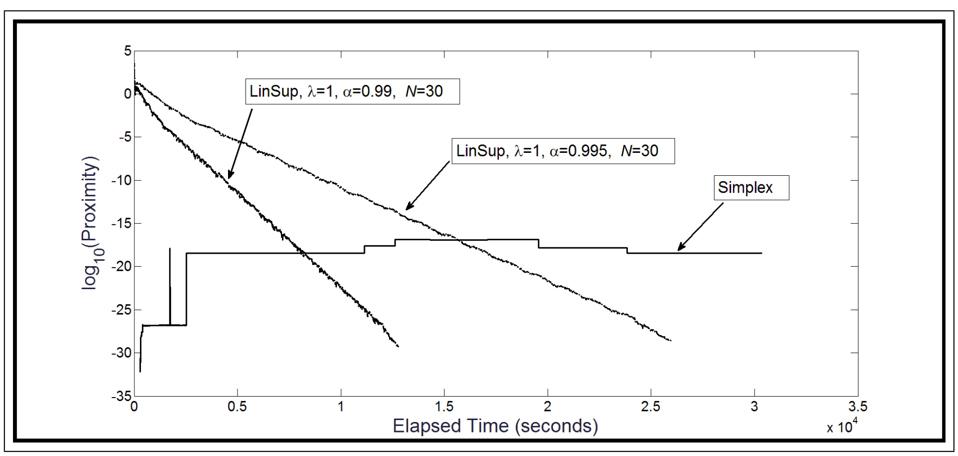

Figure 13: After 20,000 seconds, both LinSup runs would have a lower proximity than the Simplex and the one with the higher kernel value α would even have a lower linear target function value.