International Institute for Applied Systems Analysis
IIASA www.iiasa.ac.at

Augmented Simplex: A Modified and Parallel Version of Simplex Method Based on Multiple Objective and Subdifferential Optimization Approach

Wierzbicki, A.P.

IIASA Working Paper

October 1993

Working Paper

Augmented Simplex: a Modified and Parallel Version of Simplex Method Based on Multiple Objective and Subdifferential Optimization Approach

Andrzej P. Wierzbicki

WP-93-059
October 1993

IIASA International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 715210 □ Telex: 079 137 iiasa a □ Telefax: +43 2236 71313

Augmented Simplex: a Modified and Parallel Version of Simplex Method Based on Multiple Objective and Subdifferential Optimization Approach

Andrzej P. Wierzbicki

WP-93-059
October 1993

International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 715210 □ Telex: 079 137 iiasa a □ Telefax: +43 2236 71313

Foreword

This initial research for this paper was performed in the Institute of Automatic Control, Warsaw University of Technology, supported by the grant No. 3 0218 91 01 of the Committee for Scientific Research of the Republic of Poland. The Institute of Automatic Control has a cooperative research agreement with IIASA and the final version of this paper has been completed during the stay of the author at the Methodology of Decision Analysis Project. While based on the multiple criteria approach that has been intensively studied in MDA project, the paper addresses new issues of coarse-grained parallel computations.

In the perspective of parallel computations, new versions of basic optimization algorithms are needed. The paper presents a concept of such coarse-grained parallelization, based on a parametric imbedding into a family of problems or parametrically diversified algorithms. This general idea is exemplified for the case of the simplex algorithm of linear programming, where a linear optimization problem can be imbedded into a multiple-objective family which introduces diversified directions of search cutting through the interior of original admissible set. To improve the effectiveness of such algorithms, an initial phase of directional feasibility search by subdifferential optimization is added. The resulting *augmented simplex algorithm*, even without parallelization, might be competitive with interior point methods for a certain, broad class of linear programming problems. Necessary theoretical foundations, some algorithmic details and results of preliminary tests are presented.

Contents

1 Introduction. 1

2 Preliminaries. 2
- 2.1 Standard formats for linear simulation models. 2
- 2.2 Multiple-objective aspects of linear programming. 3

3 An augmented simplex method. 5
- 3.1 An augmented linear programming problem. 5
- 3.2 An example of a discretized circle. 7

4 Algorithmic aspects. 12
- 4.1 Initial feasibility phase based on subdifferentiable optimization. 12
- 4.2 A repetitive augmented simplex algorithm. 14
- 4.3 A parallel version of the augmented simplex algorithm. 16

5 Conclusions. 17

A Appendices. 19
- A.1 A subdifferential feasibility algorithm. 19
- A.2 An outline of a repetitive simplex algorithm for sequential computations. . 20
- A.3 Details of parallel version of the augmented simplex algorithm. 22

Augmented Simplex: a Modified and Parallel Version of Simplex Method Based on Multiple Objective and Subdifferential Optimization Approach

*Andrzej P. Wierzbicki**

1 Introduction.

It is well known that the simplex method of linear programming might perform slowly for problems with a large number of constraints, because it must go around the set of admissible solutions and cannot cut through the interior of this set. This is one of the reasons of the superiority of interior point algorithms using nonlinear penalty barrier functions (that have been also adapted for multi-objective linear programming, see e.g. Karmarkar, 1984, Arbel, 1992, Gondzio and Makowski, 1993). However, the property of searching in the interior of the admissible set is related not exclusively to the internal point methods. Some quasi-simplex or multiplier-type algorithms (see Gabasov and Kirillova, 1977, Makowski and Sosnowski, 1989) can also search through the interior. Moreover, as shown later in this paper, some formulations of multiple-objective linear programming introduce cuts through the interior even if the traditional simplex method is used. Thus, when treating an original single-objective problem as an example of a multi-objective one, we can improve the performance of the simplex method.

The resulting augmented simplex algorithm could be, for a certain class of problems, competitive with the internal point methods. The main advantage of the augmented simplex algorithm, however, is that it admits a natural *coarse-grain parallelization* which might result in a shortening of computing time as well as in an increased robustness of the algorithm for badly conditioned problems.

The perspective of parallel computing on many processors with large computing power — each well capable of solving problems of considerable size — has motivated recently many studies of parallel computations (see e.g. Nogi, 1986, Bertsekas and Tsitsiklis, 1989), in particular for linear programming (see e.g. Ruszczyński, 1992). Most of these studies concentrate on a decomposition of a large-scale problem in such a way that the decomposed parts can be processed parallel - that is, on using the increased computing power for solving larger problems.

*Institute of Automatic Control, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland; the research reported in this paper was supported by the grant No. 3 0218 91 01 of the Committee of Scientific Research of Poland.

There might be, however, also another direction of using additional computing power in order to increase the speed and accuracy of analyzing problems of moderate scale. Since optimization is one of basic instruments in such analysis, we can expect requirements of fast and reliable execution of many repeated optimization runs, as e.g. in multi-objective decision support systems. Thus, we shall address here the question how to use *future parallel computing power under the assumption that the analyzed models are not too complicated to be run in reasonable time on a single processor.*

In such a case, we might use coarse grain parallelization based on a *parametric imbedding* of the original problem or algorithm into a larger family - e.g. a single-objective linear programming problem can be naturally imbedded into a related multiple-objective family - and then by coordinating the process of solving problems of this family parallel in order to obtain the solution of the original problem more quickly and robustly. This general idea of coarse-grain parallelization via parametric imbedding scheme differs in its essence from the general *decomposition and coordination scheme.* Moreover, this general idea applies not only for linear, but also for nonlinear or even discrete programming. However, in this paper we address only linear programming problems.

2 Preliminaries.

2.1 Standard formats for linear simulation models.

A standard form of a linear programming problem is usually written as:

$$\underset{\boldsymbol{x}\in X_o}{\text{maximize}}(z = \boldsymbol{c}^T\boldsymbol{x}),\ X_o = \{\boldsymbol{x} \in \boldsymbol{R}^n : \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u},\ \ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \in \boldsymbol{R}^m\} \tag{1}$$

where the vector $\boldsymbol{x}$ contains, beside original decision variables, also many artificial (slack) variables and thus $n > m$ is assumed.

However, a modeler that constructs, modifies and analyses (often multi-objectively, see e.g. Wierzbicki 1992b) a linear model might find it more convenient to develop his intuitive understanding in relation to a different format that is similar to one used in (Murtagh and Saunders, 1977):

$$\text{maximize}(z = y_i),\ X_o = \{\boldsymbol{x} \in \boldsymbol{R}^n : \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u},\ \ \boldsymbol{b} \leq \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{W}y < \boldsymbol{b} + \boldsymbol{r} \in \boldsymbol{R}^m\} \tag{2}$$

In this format, the vector $\boldsymbol{x}$ denotes only some *essential or structural decision variables* while the vector $\boldsymbol{y}$ includes all *outcome and intermediate variables* used e.g. to express additional constraints. If the model is complicated, then $n << m$. The matrix $\boldsymbol{W}$ reflects the fact that the modeler might introduce various intermediate outcome variables. Usually, $(\boldsymbol{I} - \boldsymbol{W})^{-1}$ exists and $\boldsymbol{y}$ can be determined as a direct function of $\boldsymbol{x}$, by substituting $\boldsymbol{y} = (\boldsymbol{I} - \boldsymbol{W})^{-1}\boldsymbol{A}\boldsymbol{x}$ and $\boldsymbol{A} := (\boldsymbol{I} - \boldsymbol{W})^{-1}\boldsymbol{A}$; but this should be done by a preprocessing software, not by the modeler. Such a preprocessor can also perform many other transformations resulting in a simplified format similar to (1):

$$\text{maximize}\,(z = \boldsymbol{c}^T\boldsymbol{x}),\ X_o = \{\boldsymbol{x} \in \boldsymbol{R}^n :\ \boldsymbol{l} < \boldsymbol{x} < \boldsymbol{u},\ \boldsymbol{b} < \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} < \boldsymbol{b} + \boldsymbol{r} \in \boldsymbol{R}^m\} \tag{3}$$

where $\text{Int}\, X_o \neq \emptyset$ and $l_j < u_j,\ c_j > 0\ \forall j = 1,\ldots n,\ r_i > 0\ \forall i = 1,\ldots m$ might be assumed as the result of the preprocessing.

We stress that $\boldsymbol{x}$ *denotes here only essential decision variables.* For example, the iterates of an internal point method that are in the interior of the bounds on $\boldsymbol{x}$ in formulation (1) correspond rather to internal points of X_o in formulation (3). The assumption that $c_j > 0$ is made for exposition purposes only, without loss of generality.

The class of linear problems considered in this paper is such that *the number n of essential decision variables is relatively small* - say, comparable to the number of available processors in a parallel computer - *while the number m of inequality constraints might be very large.* Many practical examples and many subclasses of linear and discrete programming - e.g. semi-infinite LP - belong to this class. The selection of essential decision variables might be not unique for a given problem; although it is usually implied by the original problem formulation, we can also aggregate parts of cost function with original decision variables into new variables that can be interpreted as essential ones.

2.2 Multiple-objective aspects of linear programming.

Consider the following formulation of a multiple-objective linear programming problem:

$$\text{"}\underset{\boldsymbol{x}\in X_o}{\text{maximize}}\text{"}(\boldsymbol{q} = \boldsymbol{C}\boldsymbol{x} \in \boldsymbol{R}^p),\ X_o = \{\boldsymbol{x} \in \boldsymbol{R}^n : \boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u},\ \boldsymbol{b} \le \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b} + \boldsymbol{r} \in \boldsymbol{R}^m\} \quad (4)$$

where, usually, the matrix $\boldsymbol{C}$ is defined by selecting $q_{\hat{i}} = y_{\hat{i}}$ for some $\hat{i} \in \hat{I}$; however, there might be several interpretations of "maximization". Generally, we can define any *positive* (closed, convex, pointed) *cone* $D \subset \boldsymbol{R}^p$, a corresponding *strictly positive cone* $\tilde{D}$ and define the *set of efficient solutions* to (4) w.r.t. (with respect to) this cone:

$$\hat{X}_o = \{\hat{\boldsymbol{x}} \in X_o : (\hat{\boldsymbol{q}} + \tilde{D}) \cup Q_o = \emptyset \text{ where } \hat{\boldsymbol{q}} = \boldsymbol{C}\hat{\boldsymbol{x}},\ Q_o = \boldsymbol{C}X_o\},\ \text{with } \hat{Q}_o = \boldsymbol{C}\hat{X}_o \quad (5)$$

If $D = \boldsymbol{R}^p_+$, all criteria or objectives $q_{\hat{i}}$ are to be maximized, then by choosing $\tilde{D} = D \setminus \{0\}$ we obtain the set $\hat{X}_o$ of Pareto-optimal solutions, while $\hat{Q}_o$ is called the Pareto frontier of the attainable objective set Q_o . If we choose $\tilde{D} = \text{Int}\,\boldsymbol{R}^p_+$, then we obtain larger sets of weakly Pareto-optimal solutions and attainable objectives. The problem of finding Pareto-optimal solutions to linear programming models has been investigated extensively - see e.g. (Yu and Zeleny (1975), Gal (1977), Steuer (1986)). Most of such approaches are related to a parameterization of $\hat{X}_o$ or $\hat{Q}_o$ as a set of maxima of *scalarizing functions,* e.g. in the form of a weighted sum:

$$s_1(\boldsymbol{q}, \boldsymbol{\alpha}) = \sum_{\hat{i}=1}^{p} \alpha_{\hat{i}} q_{\hat{i}}$$

It is well known that, if $\boldsymbol{\alpha} \in \text{Int}\,\boldsymbol{R}^p_+$, then all maxima of $s_1(\boldsymbol{C}\boldsymbol{x}, \boldsymbol{\alpha})$ w.r.t. $\boldsymbol{x} \in X_o$ are Pareto-optimal. Conversely, since X_o and Q_o are convex, polyhedral sets, for any Pareto-optimal $\hat{\boldsymbol{q}} = \boldsymbol{C}\hat{\boldsymbol{x}}$ there exists an $\hat{\boldsymbol{\alpha}} \in \text{Int}\,\boldsymbol{R}^p_+$ such that $\hat{\boldsymbol{x}}$ maximizes $s_1(\boldsymbol{C}\boldsymbol{x}, \hat{\boldsymbol{\alpha}})$ w.r.t. $\boldsymbol{x} \in X_o$.

However, there are also other cones D and $\tilde{D}$ that might express various requirements of mixed maximization and minimization or even *stabilization of objectives* $q_{\hat{i}}$, and other forms of scalarizing functions that are applicable also for non-convex and discrete cases and express then the additional requirement of *proper* Pareto-optimality or *proper efficiency*

(with either bounded or even a priori bounded trade-off coefficients, see e.g. Wierzbicki, 1986, Seo and Sakawa, 1988, Lewandowski and Wierzbicki, 1989). It might seem that these other forms are not needed in the case of linear models (where all Pareto-optimal points are proper); but these alternative approaches have some desirable properties even in this case.

A general approach to the parameterization of efficient solutions is related to the following lemma (Wierzbicki, 1986, 1992a). To simplify notation, we substitute $r(\boldsymbol{q}) := s(\boldsymbol{q}, \boldsymbol{\alpha})$ in this lemma.

Lemma 1 *Suppose X_o is compact and consider a continuous vector objective function $\boldsymbol{f} : X_o \to \boldsymbol{R}^p, \boldsymbol{q} = \boldsymbol{f}(\boldsymbol{x})$. Given a strictly positive cone $\tilde{D}$, if a continuous function $r : \boldsymbol{R}^p \to \boldsymbol{R}^1$ is strictly monotone w.r.t. this cone ($\boldsymbol{q}' \in \boldsymbol{q}'' + \tilde{D} \Rightarrow r(\boldsymbol{q}') > r(\boldsymbol{q}'')$), then each maximum of $r(\boldsymbol{f}(\boldsymbol{x}))$ w.r.t. $\boldsymbol{x} \in X_o$ is efficient w.r.t. $\tilde{D}$. Conversely, if a continuous function $r : \boldsymbol{R}^p \to \boldsymbol{R}^1$ strictly separates the sets $Q_o = \boldsymbol{f}(X_o)$ and $\hat{\boldsymbol{q}} + \tilde{D}$ for any $\hat{\boldsymbol{q}} \in Q_o$ (if $r(\boldsymbol{q}') > 0 \; \forall \boldsymbol{q}' \in \hat{\boldsymbol{q}} + \tilde{D}$ and $r(\boldsymbol{q}'') \leq 0 \; \forall \boldsymbol{q}'' \in Q_o$), then this property is equivalent to the three following statements:*

a) $\hat{\boldsymbol{q}} = \boldsymbol{f}(\hat{\boldsymbol{x}})$ *maximizes* $r(\boldsymbol{q})$ *w.r.t.* $\boldsymbol{q} \in Q_o$ ($\hat{\boldsymbol{x}}$ *maximizes* $r(\boldsymbol{f}(\boldsymbol{x}))$ *w.r.t.* $\boldsymbol{x} \in X_o$);

b) $\hat{\boldsymbol{q}} \in \hat{Q}_o$ *and* $\hat{\boldsymbol{x}} \in \hat{X}_o$, *they are efficient w.r.t.* $\tilde{D}$;

c) $r(\boldsymbol{q}') > r(\hat{\boldsymbol{q}}) \; \forall \boldsymbol{q}' \in \hat{\boldsymbol{q}} + \tilde{D}$, *the function* r *is at least locally strictly monotone.*

Thus, a parameterization of $\hat{Q}_o$ and $\hat{X}_o$ in the sense of both sufficient and necessary conditions of efficiency might be obtained by choosing a suitably parameterized, strictly monotone scalarizing function that separates Q_o and $\hat{\boldsymbol{q}} + \tilde{D}$. For linear models, where $\hat{Q}_o$ and $\hat{X}_o$ are convex polyhedral sets, the choice of the weighted sum $s_1(\boldsymbol{q}, \boldsymbol{\alpha}) = r(\boldsymbol{q})$ might seem the most natural.

But a piece-wise linear separating function might approximate the positive cone better. A piece-wise linear function that has all the properties needed by Lemma 1 — while, not knowing in advance what is $\hat{\boldsymbol{q}}$, we use initially any $\bar{\boldsymbol{q}}$ as a *reference point* in objective space — might be written as follows:

$$s_2(\boldsymbol{q}, \bar{\boldsymbol{q}}, \boldsymbol{\alpha}) = \min_{i=1,\dots p} \alpha_i(q_i - \bar{q}_i) + \epsilon \sum_{i=1}^{p} \alpha_i(q_i - \bar{q}_i) \tag{6a}$$

By maximizing this function, a Pareto-optimal point $\hat{\boldsymbol{q}}$ is obtained; then the reference point $\bar{\boldsymbol{q}}$ might be shifted to $\hat{\boldsymbol{q}}$. The parameter $\epsilon > 0$ is used in this function in order that it separates the set Q_o from the cone $\bar{\boldsymbol{q}} + \operatorname{Int} D_\epsilon$, where D_ϵ is a slightly broader cone than $\boldsymbol{R}^p_+$ that characterizes properly Pareto-optimal outcomes *with a prior bound* $1 + 1/\epsilon$ *on trade-off coefficients* (see e.g. Wierzbicki 1986, 1992a). However, in this paper we do not need such property and thus a simpler form with $\epsilon = 0$ will be used:

$$s_2(q, \bar{q}, \boldsymbol{\alpha}) = \min_{i=1,\dots p} \alpha_i(q_i - \bar{q}_i) \tag{6b}$$

When this function is used to characterize the set of (weakly) Pareto- optimal decisions and outcomes, then the controlling parameter of such characterization is the reference point vector $\bar{\boldsymbol{q}}$ and the weighting coefficients $\alpha_i > 0$ play an auxiliary role. Here, however, we shall reverse the roles of these parameters; note that the weighting coefficients can

be also expressed as scaling factors or directional coefficients $d_i = 1/\alpha_i$, with $\boldsymbol{d} \in \boldsymbol{R}^p$ interpreted as a direction of search.

If Q_o and X_o are convex, polyhedral sets as in (4), (5), then the maximization of $s_2(\boldsymbol{C}\boldsymbol{x}, \bar{\boldsymbol{q}}, \boldsymbol{\alpha})$ over $\boldsymbol{x} \in X_o$ can be equivalently written as a modified linear programming problem:

$$\max_{(x,\tau)\in XT_o} \tau; \quad XT_o = \{(\boldsymbol{x}, \tau) \in \boldsymbol{R}^{n+1} : \boldsymbol{x} \in X_o, \ \alpha_i \bar{q}\hat{i} + \tau \leq \alpha_i \boldsymbol{c}^{(\hat{i})T}\boldsymbol{x}, \ \text{all} \ i = 1, \ldots p\} \quad (7)$$

where $\boldsymbol{c}^{(\hat{i})T}$ are the rows of the matrix $\boldsymbol{C}$, $q_i = \boldsymbol{c}^{(\hat{i})T}\boldsymbol{x}$. This modified formulation has been used, for example, in the programming package for multiple-objective analysis and decision support DIDAS-L for linear models (see Rogowski et al., 1989; similar forms have been used earlier by many authors, see e.g. Wierzbicki, 1980, Nakayama and Sawaragi, 1983, Steuer and Choo, 1983).

The modified problem has several specific properties. Its admissible basic solutions include but are not necessarily vertices of the set X_o ; its optimal solution might be any (weakly) Pareto optimal point $\hat{\boldsymbol{x}} \in \hat{X}_o$, also located on a facet or edge of the boundary of X_o . This property was sometimes considered a disadvantage — see comments to a similar approach of Ecker and Kuada in Steuer, 1986 — but it is possible to use it to advantage. The modified problem might be better conditioned than the original one, since the additional constraints are orthogonal to each other. Another valuable property, observed in applications of the package DIDAS-L mentioned above, is that in certain cases the modified problem requires less simplex iterations than a comparable scalar optimization problem; this occurs because the modified formulation introduces *an additional edge of the simplex that cuts through the interior of the original one.*

3 An augmented simplex method.

3.1 An augmented linear programming problem.

Suppose we solve problem (3): maximize a scalar function $q = \boldsymbol{c}^T\boldsymbol{x}$ with respect to $\boldsymbol{x} \in X_o$, and let $c_j > 0$ for all $j = 1, \ldots n$. Then an augmented linear programming problem can be written as:

$$\max_{\boldsymbol{x}\in X_o} s_\rho(\boldsymbol{x}, \boldsymbol{d}, \bar{\boldsymbol{x}}); \quad s_\rho(\boldsymbol{x}, \boldsymbol{d}, \bar{\boldsymbol{x}}) = \rho \boldsymbol{c}^T\boldsymbol{x} + (1-\rho) \min_{j=1,\ldots n} (x_j - \bar{x}_j)/d_j \quad (8a)$$

where $\boldsymbol{d}$ can be interpreted as a direction in the solution space, while $d_j = 1/\alpha_j > 0$ can be also interpreted as inverse weighting coefficients for the problem of maximizing multi-objectively all components x_j as in (6b) with $p = n$ and $x_j = q_j$; $\bar{\boldsymbol{x}} = \bar{\boldsymbol{q}}$ is then a reference point in the solution space.

Since $c_j > 0$ for all j, we know (e.g. from Lemma 1) that *the solution* $\hat{\boldsymbol{x}}$ *to the original problem of maximizing* $\boldsymbol{c}^T\boldsymbol{x}$ w.r.t. $\boldsymbol{x} \in X_o$ *belongs to the Pareto boundary* $\hat{X}_o$ *for the problem of maximizing* all components x_j multi- objectively. Thus, the augmented problem can be solved in several phases with changing coefficient $\rho \in \{0, 1\}$:

- an initial *feasibility phase* might be needed;
- in the middle *multi-objective phase*, $\rho = 0$ until a point $\tilde{\boldsymbol{x}} \in \hat{X}_o$ is found;

- in the final *single-objective phase*, $\rho = 1$ until the solution $\hat{\boldsymbol{x}}$ to the original problem is found.

Note that the switch between phases corresponds only to a change of maximized function in the equivalent linear programming problem stated similarly as (7):

$$\max_{(\boldsymbol{x},\tau)\in XT_o}(\tau \text{ or } \boldsymbol{c}^T\boldsymbol{x}); \quad XT_o = \{(\boldsymbol{x},\tau) \in \boldsymbol{R}^{n+1} : \boldsymbol{x} \in X_o,\ \bar{x}_j \le x_j - d_j\tau,\ j = 1,\dots n\} \quad (8b)$$

thus the second phase can be started from the optimal basic solution of the first phase.

The properties of the augmented linear programming problem are summarized in the following Lemma.

Lemma 2 *For the problem (8b) with X_o defined as in (3), let $\boldsymbol{c}, \boldsymbol{d} \in \text{Int}\, \boldsymbol{R}^n_+$ and denote:*

$$T(\bar{\boldsymbol{x}}, \boldsymbol{d}) = \{\boldsymbol{x} \in \boldsymbol{R}^p : \boldsymbol{x} = \bar{\boldsymbol{x}} + \tau\boldsymbol{d},\ \ \tau \in \boldsymbol{R}^1\} \quad (8c)$$

If $T(\bar{\boldsymbol{x}}, \boldsymbol{d}) \cap X_o \neq \emptyset$, then for any $\boldsymbol{x} \in T(\bar{\boldsymbol{x}}, \boldsymbol{d}) \cap X_o$ an admissible basic solution to (8a) can be determined; in one simplex iteration, this solution can be transformed to such that maximizes τ and $\boldsymbol{c}^T\boldsymbol{x}$ w.r.t. $\boldsymbol{x} \in T(\bar{\boldsymbol{x}}, \boldsymbol{d}) \cap X_o$.

Proof. Consider the way of transforming problem (8b) into the standard form (1). We add two vectors of slack variables: one for all original constraints $b_i \le \boldsymbol{a}^{iT}\boldsymbol{x} \le b_i + r_i$, $\boldsymbol{s} \in \boldsymbol{R}^m$ with $0 \le \boldsymbol{s} \le \boldsymbol{r}$, and another for additional constraints $\bar{x}_j + d_j\tau \le \bar{x}_j$, $\boldsymbol{v} \in \boldsymbol{R}^n$ with $0 \le \boldsymbol{v}$. Thus we obtain the set of $n + m$ equations for $2n + m + 1$ variables $\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{s}, \tau$:

$$\boldsymbol{x} - \boldsymbol{v} - \tau\boldsymbol{d} = \bar{\boldsymbol{x}};\ \ \boldsymbol{A}\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{b} \quad (9a)$$

Substitute $\boldsymbol{x}$ computed from the first n equations into the remaining m equations to obtain:

$$\boldsymbol{x} - \boldsymbol{v} - \tau\boldsymbol{d} = \bar{\boldsymbol{x}};\ \ \boldsymbol{s} + \boldsymbol{A}\boldsymbol{v} + \tau\boldsymbol{A}\boldsymbol{d} = \boldsymbol{b} - \boldsymbol{A}\bar{\boldsymbol{x}} \quad (9b)$$

If $\bar{\boldsymbol{x}} \in X_o$, then $\boldsymbol{x} = \bar{\boldsymbol{x}}$ and $\boldsymbol{s} = \bar{\boldsymbol{s}} = \boldsymbol{b} - \boldsymbol{A}\bar{\boldsymbol{x}}$ can be selected as starting values of basic variables, with a corresponding unit basic matrix, while $\boldsymbol{v} = 0$ and $\tau = 0$ are not in the basis. Note that $T(\bar{\boldsymbol{x}}, d) = T(\boldsymbol{x}, \boldsymbol{d})$ for any $\boldsymbol{x} \in T(\bar{\boldsymbol{x}}, \boldsymbol{d})$; therefore, if $\bar{\boldsymbol{x}} \notin X_o$ but $T(\bar{\boldsymbol{x}}, d) \cap X_o \neq \emptyset$, we can select any $\boldsymbol{x} \in T(\bar{\boldsymbol{x}}, d) \cap X_o$ and reset $\bar{\boldsymbol{x}} := \boldsymbol{x}$ thus obtaining an admissible starting point.

Because the method changes objective functions between phases, consider two reduced cost vectors — corresponding to these phases — at this starting point. The reduced cost vector for the multi-objective phase with maximized function τ has clearly only one non-zero coordinate 1 corresponding to τ. The reduced cost vector for the single-objective phase with maximized function $\boldsymbol{c}^T\boldsymbol{x}$ results from:

$$\boldsymbol{c}^T\boldsymbol{x} = \boldsymbol{c}^T\boldsymbol{v} + \tau\boldsymbol{c}^T\boldsymbol{d} + \boldsymbol{c}^T\bar{\boldsymbol{x}}$$

Since $\boldsymbol{x}$ and $\boldsymbol{s}$ are in the basis, the corresponding reduced cost vector is $(0\ 0\ \boldsymbol{c}^T\ \boldsymbol{c}^T\boldsymbol{d})^T$. In the first iteration of the multi-objective phase, τ will be selected as the variable that

enters new basis (since τ is actually unbounded, it must stay in the basis in further iterations, i.e. be omitted when selecting the variables that leave the basis). The variable that leaves the basis in the first iteration is either a slack variable s_i or a variable x_j that first attains its bounds when $\boldsymbol{x}$ is changing along the direction $\boldsymbol{d}$, $\boldsymbol{x} = \bar{\boldsymbol{x}} + \tau \boldsymbol{d}$ - since $\boldsymbol{v} = 0$ at the starting point.

Thus, the boundary of X_o is attained in the first iteration at a point $\boldsymbol{x} \in T(\bar{\boldsymbol{x}}, \boldsymbol{d}) \cap X_o$ that maximizes τ; this point maximizes also $\boldsymbol{c}^T\boldsymbol{x}$ in $T(\bar{\boldsymbol{x}}, \boldsymbol{d}) \cap X_o$, since $\boldsymbol{c}^T\boldsymbol{d} > 0$. This concludes the proof.

Although an efficient implementation should be naturally based on an appropriate product form with good numerical stability and sparsity preservation features, the classical simplex tableau for this problem might be used to illustrate some aspects of the method. In the following tableau, two last rows represent the alternative reduced cost vectors:

$$\left[\begin{array}{cccc|c} \boldsymbol{I} & 0 & -\boldsymbol{I} & -\boldsymbol{d} & \bar{\boldsymbol{x}} \\ 0 & \boldsymbol{I} & \boldsymbol{A} & \boldsymbol{Ad} & \bar{\boldsymbol{s}} \\ -- & -- & -- & -- & -- \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \boldsymbol{c} & \boldsymbol{c}^T\boldsymbol{d} & \boldsymbol{c}^T\bar{\boldsymbol{x}} \\ (\boldsymbol{x}) & (\boldsymbol{s}) & (\boldsymbol{v}) & (\tau) & \end{array}\right] \begin{array}{c} (\boldsymbol{x}) \\ (\boldsymbol{s}) \\ \\ \\ \\ \end{array} \qquad (9b)$$

Note that it is advantageous to choose $\bar{\boldsymbol{x}} \in \operatorname{Int} X_o$ — if $\bar{\boldsymbol{x}} \notin \operatorname{Int} X_o$ and $\bar{\boldsymbol{x}}$ is in such boundary of X_o that a further increase of τ along the direction $\boldsymbol{x} = \bar{\boldsymbol{x}} + \tau \boldsymbol{d}$ is impossible, then the initial basis is degenerate. Such a point is always attained after the first iteration, but with a nondegenerate basis if $\bar{\boldsymbol{x}} \in \operatorname{Int} X_o$.

Such a point might not yet belong to the Pareto boundary $\hat{X}$, since directional maximization does not necessarily imply Pareto-optimality (see e.g. Wierzbicki 1986). However, after a finite number of iterations the maximum of τ in XT_o will be found at some point $\tilde{\boldsymbol{x}}$, and the maximization of $\boldsymbol{c}^T\boldsymbol{x}$ might start from the basis corresponding to this point. The number of simplex iterations in the multi-objective phase might be very small — one, if $T(\bar{\boldsymbol{x}}, \boldsymbol{d})$ intersects $\hat{X}_o$ — or large — when $T(\bar{\boldsymbol{x}}, \boldsymbol{d})$ intersects the boundary of X_o far away from $\hat{X}_o$. Usually, such a point $\tilde{\boldsymbol{x}}$ is not a vertex of X_o , it might belong to a facet of X_o . Vertices of X_o are attained after some iterations in the final single-objective phase, when continuing to maximize $\boldsymbol{c}^T\boldsymbol{x}$.

Thus, the augmented simplex formulation has some similarities with interior point methods: it might start from an interior point and cut through the interior of X_o in the first iteration. Additionally, since any $\boldsymbol{d} \in \operatorname{Int} \boldsymbol{R}^n_+$ might be selected, the original problem of maximizing $\boldsymbol{c}^T\boldsymbol{x}$ w.r.t. $\boldsymbol{x} \in X_o$ is em parametrically imbedded into a family of problems - or rather algorithms with parameters $\bar{\boldsymbol{x}}, \boldsymbol{d}$.

3.2 An example of a discretized circle.

As a way of illustrating theoretical considerations and preliminary testing more detailed algorithmic suggestions, a widely used example was selected: maximize a linear function w.r.t. $\boldsymbol{x} \in C_o$, where C_o is a circle approximated by many linear constraints. For example, the circle $(\boldsymbol{x}_1 - 2)^2 + x_2^2 \leq 4$ through a substitution $x_1 = 2(1 + cos\beta)$, $x_2 = 2sin\beta$, $\beta \in [0; \pi]$ can be approximated by:

$$C_o = \{\boldsymbol{x} \in \boldsymbol{R}^2 : \; b_i \leq y_i = a_{1i}x_1 + a_{21}x_2 \leq b_i + r_i, \; i = 1, \ldots m\} \qquad (10a)$$

where:

$$b_i = 2(cos(\beta_i + \Delta\beta) - cos\Delta\beta);\ r_i = 4cos\Delta\beta \tag{10b}$$

$$a_{1i} = cos(\beta_i + \Delta\beta),\ a_{2i} = sin(\beta_i + \Delta\beta);\ \beta_i = i\pi/m,\ \Delta\beta = \pi/2m \tag{10c}$$

while the step of changing β is actually $2\Delta\beta = \pi/m$.

Testing results for this rather special example should be treated with care: the advantages of the augmented simplex method would be exaggerated if a very large m were chosen, hence a moderate $m = 90$ was selected. Additional constraints were selected as $\boldsymbol{x} \in B_o = \{\boldsymbol{x} \in \boldsymbol{R}^2 : \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}\}$, $X_o = C_o \cap B_o$, where $\boldsymbol{l}$ and $\boldsymbol{u}$ might be changed during various experiments with this example. In the maximized function $z = \boldsymbol{c}^T\boldsymbol{x}, \boldsymbol{c} = (3\ \ 2)^T$ was typically assumed. Thus, the problem to be solved in this example is:

$$\underset{\boldsymbol{x} \in X_o}{\text{maximize}}(q_o = c_1x_1 + c_2x_2);\ X_o = \{\boldsymbol{x} \in \boldsymbol{R}^2 :\ \boldsymbol{x} \in C_o,\ \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}\} \tag{11}$$

This problem was imbedded into a multi-objective family with the help of multiple-criteria decision support system DIDAS-L. This public domain software [1] was slightly modified for such experiments. DIDAS-L determines automatically so called *utopia points*; in order to increase the freedom of experimentation, user-supplied utopia points were admitted. The simplex solver in DIDAS-L does not exploit the transformation (9b); therefore, $n+1 = 3$ iterations instead of 1 iteration are needed to cut through the interior of X_o . Even with this drawback, the results of preliminary experiments with this example presented in Table 1 are quite promising.

Table 1. Results of experiments with augmented problem, $c = (3\ \ 2)^T$, $u = (5\ \ 3)^T$

Bounds		Optim. type	Separate			Augmented								
l_1	l_2		max q_o	max x_1	max x_2	$d_1\ d_2$ 9 1	$d_1\ d_2$ 8 2	$d_1\ d_2$ 7 3	$d_1\ d_2$ 6 4	$d_1\ d_2$ 5 5	$d_1\ d_2$ 4 6	$d_1\ d_2$ 3 7	$d_1\ d_2$ 2 8	$d_1\ d_2$ 1 9
0	-2	Number of it.	86	67	68	39 +16	34 +16	31 +16	29 +8	29 +6	29 +20	34 +29	43 +29	56 +29
		q_o	13.2	12.0	10.0	12.0	12.0	12.0	12.9	13.1	11.6	10.0	10.0	10.0
		x_1	3.66	4.0	2.0	4.0	4.0	4.0	3.91	3.41	2.6	2.0	2.0	2.0
		x_2	1.12	0.0	2.0	0.0	0.0	0.0	0.6	1.41	1.91	2.0	2.0	2.0
1	-1	Number of it.	21	5	19	5 +16	5 +16	3 +13	3 +5	3 +6	3 +16	3 +25	7 +29	13 +29
		q_o	13.2	12.0	10.0	12.0	12.0	12.0	12.9	13.1	11.6	10.0	10.0	10.0
		x_1	3.66	4.0	2.0	4.0	4.0	3.98	3.8	3.41	2.87	2.28	2.0	2.0
		x_2	1.12	0.0	2.0	0.0	0.0	0.28	0.87	1.41	1.8	1.98	2.0	2.0

In Table 1, separate optimization means single-objective maximization of $q_o = \boldsymbol{c}^T\boldsymbol{x}$ or of x_1 or x_2 , and the corresponding number of simplex iterations and the optimal values of $x_1 = \hat{x}_1$, $x_2 = \hat{x}_2$, $q_o = \boldsymbol{c}^T\hat{\boldsymbol{x}}$ are given. For the case of infeasible $\boldsymbol{l} = (0\ \ -2)^T$, the numbers of simplex iterations include the initial feasibility phase, which takes between 26 and 45 iterations; for the case of feasible $\boldsymbol{l} = (1\ \ -1)^T$, no feasibility phase iterations are

[1] Described shortly e.g. in Rogowski et al., 1989; full information and software can be obtained from Dr Marek Makowski at the International Institute for Applied Systems Analysis in Laxenburg, Austria

needed. The iterations started always from $\bar{\boldsymbol{x}} = \boldsymbol{l}$ (if they started from $\bar{\boldsymbol{x}} = \boldsymbol{u}$, then 33 iterations would be needed for the maximization of q_o , even with feasible $\boldsymbol{l} = (1 \ -1)^T$). For the augmented formulation, instead of the optimal values, the values $x_1 = \tilde{x}_1$, $x_2 = \tilde{x}_2$, $q_o = \boldsymbol{c}^T\tilde{\boldsymbol{x}}$ after the multiple-objective phase are given in Table 1. The needed numbers of iterations are split into two parts: the iterations in the initial phase (including a possible feasibility phase) plus the iterations in the final single-objective phase.

Clearly, we cannot assume that we know what direction $\boldsymbol{d} = (d_1 \ d_2)^T$ would result in going directly from $\bar{\boldsymbol{x}}$ to the optimal solution ($\hat{\boldsymbol{x}} = (3.66 \ 1.12)^T$ in this case). However, some reasonable directions - such as choosing $\boldsymbol{d} = \boldsymbol{u} - \boldsymbol{l}$ (which after rescaling corresponds to $\boldsymbol{d} = (5 \ 5)^T$ in Table 1) or $\boldsymbol{d} = \boldsymbol{c}$ (which corresponds to $\boldsymbol{d} = (6 \ 4)^T$ in Table 1) - result in shortening the computations more than twice. For most cases in Table 1, the total number of iterations in the augmented formulation does not exceed the number of iterations needed to maximize q_o separately - except in some cases, where the maximum of the scalarizing function in multi-objective phase is located on a flat part of this function, see Fig. 1. These results suggest a concept of parallelization of computations using the

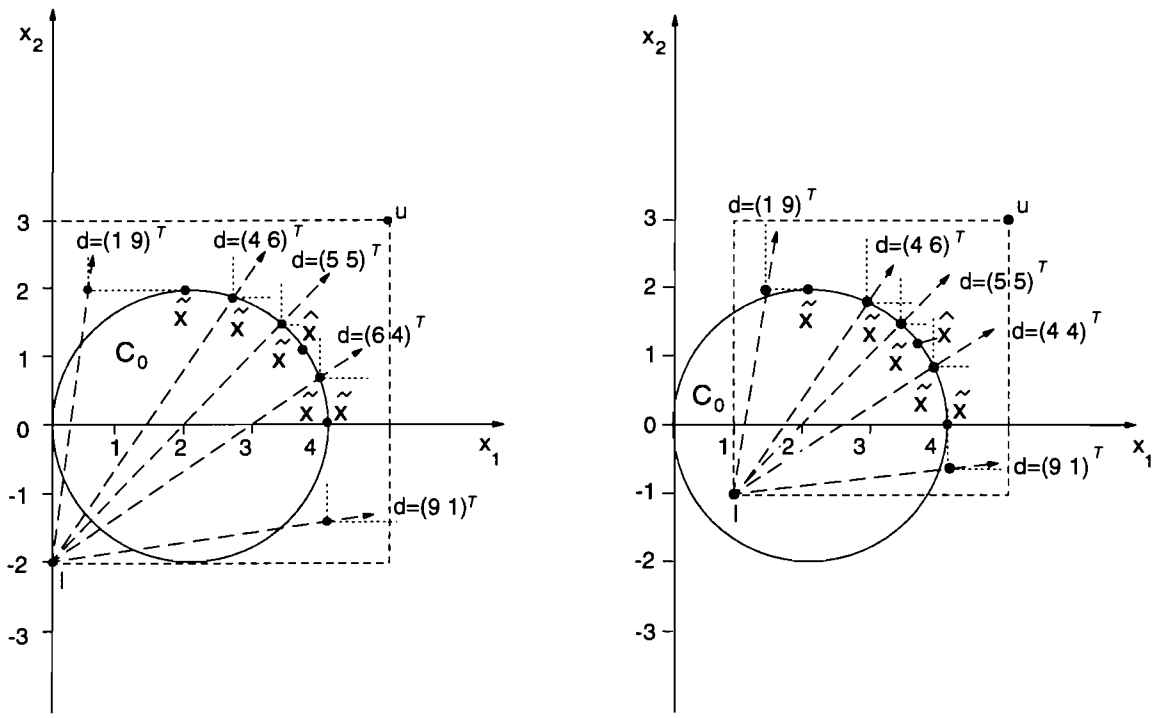

Figure 1: Illustration of experiment results from Table 1.
a) Infeasible $\bar{\boldsymbol{x}} = \boldsymbol{l}$,
b) Feasible $\bar{\boldsymbol{x}} = \boldsymbol{l}$.

augmented formulation. Suppose we employ a number P of processors, where $P > n$ in this simple example. One of the processors starts to solve the original problem of maximizing q_o ; others solve the multiple-objective augmented problems for various $\boldsymbol{d}$ and then start the single-objective phase. The iterations in all processors are stopped if in a number of them — say, n — either the optimal solution is found or a given number κ_o of iterations of the single-objective phase is performed. This stops *a large iteration*; from the results obtained, estimates $\bar{\boldsymbol{x}}$ and $\bar{\bar{\boldsymbol{x}}}$ of a *relative lower bound and upper bound* of the solution $\hat{\boldsymbol{x}}$ of the original problem are determined. *Such estimates do not serve as*

modified $\boldsymbol{l}, \boldsymbol{u}$ *which remain unchanged*, but as data used to determine the initial point in the next large iteration which starts e.g. from the new $\bar{\boldsymbol{x}}$ (a *relative lower bound strategy*) or from the new $\bar{\bar{\boldsymbol{x}}}$ (a *relative upper bound strategy*) and possibly the directions $\boldsymbol{d}$. This parallelization concept is rather strongly coarse-grained and thus intended either for distributed computing or for future computers with many powerful processors in a scalably parallel structure [2].

For the simple example with $n = 2$ considered here, the way of determining a new relative lower bound estimate $\bar{\boldsymbol{x}}$ is natural: just take the highest values of x_j obtained from various processors which would further increase if the single-objective phase would be continued. This concept of parallelization was simulated in the DIDAS-L system (using only one processor repetitively) for the simple example of discretized circle. In the following tables, large iterations are numbered by $k = 1, \ldots$ and κ denotes the number of simplex iterations (including $\kappa_o = 1$ additional) inside a large one. Other conditions of these experiments are as in Table 1: $\boldsymbol{c} = (3\ 2)^T$, $\boldsymbol{u} = (5\ 3)^T$, starting with (lower bound strategy) $\bar{\boldsymbol{x}} = \boldsymbol{l}$ which was either infeasible, $\boldsymbol{l} = (0\ -2)^T$, or feasible, $\boldsymbol{l} = (1\ -1)^T$.

The note "fail" in Table 2a means that a "processor" failed either to find optimal solution or to finish the multiple-criteria phase, thus its results were disregarded. As in Table 1, the numbers of simplex iterations needed to finish the multiple-objective phase and the values $x_1 = \tilde{x}_1$, $x_2 = \tilde{x}_2$, $q_o = \boldsymbol{c}^T\tilde{\boldsymbol{x}}$ after this phase are indicated. The first "processor" failed to find the optimal solution also in the second large iteration, but it was found by one simplex iteration of the single-objective phase in the last two "processors".

Table 2a. Results of simulated parallelization experiments, P=10 "processors", starting $\bar{x} = l = (0\ -2)^T$

Large iterat.	Optim. type	Separ. max q_o	Augmented $d_1\ d_2$ 9 1	$d_1\ d_2$ 8 2	$d_1\ d_2$ 7 3	$d_1\ d_2$ 6 4	$d_1\ d_2$ 5 5	$d_1\ d_2$ 4 6	$d_1\ d_2$ 3 7	$d_1\ d_2$ 2 8	$d_1\ d_2$ 1 9
k=1	No. of iterat.	fail	fail	fail	fail	29	29	29	fail	fail	fail
$\bar{x}_1 = 0$	q_o	–	–	–	–	12.9	13.1	11.6	–	–	–
$\bar{x}_2 = -2$	x_1	–	–	–	–	3.9	**3.61**	2.6	–	–	–
$\kappa = 30$	x_2	–	–	–	–	**0.61**	1.41	1.96	–	–	–
k=2	No. of iterat.	fail	3	3	3	3	3	3	3	3	3
$\bar{x}_1 = 3.61$	q_o	–	12.9	13.0	13.0	13.1	13.1	13.2	13.2	13.2	13.2
$\bar{x}_2 = 0.61$	x_1	–	3.9	3.88	3.87	3.85	3.82	3.79	3.75	**3.71**	**3.66**
$\kappa = 4$	x_2	–	0.63	0.67	0.71	0.76	0.82	0.89	0.96	**1.04**	**1.12**

Total length 34 iterations, compared to 86 iterations without parallelization

Similar experiments were performed also for $P = 5$, resulting in the total length of 42 iterations compared to 86 without parallelization. Note that the computational effort for coordination and communications between "processors" can be practically disregarded, since the parallelization is coarse-grained; the additional effort for coordination would be smaller than for one simplex iteration.

Similar results are obtained when the experiment starts with a feasible $\bar{\boldsymbol{x}} = \boldsymbol{l} = (1\ -1)^T$, see Table 2b.

[2]The author is indebted to Dr Tatsuo Nogi from Kyoto University for stimulating discussions on future architectures of parallel computers.

Table 2b. Results of simulated parallelization experiments,
P=10 "processors", starting $\bar{x} = l = (1\ -1)^T$

Large iterat.	Optim type	Sep1 max q_o	Augmented $d_1\ d_2$ 9 1	$d_1\ d_2$ 8 2	$d_1\ d_2$ 7 3	$d_1\ d_2$ 6 4	$d_1\ d_2$ 5 5	$d_1\ d_2$ 4 6	$d_1\ d_2$ 3 7	$d_1\ d_2$ 2 8	$d_1\ d_2$ 1 9
k=1	No. of iterat.	fail	fail	fail	3	3	3	3	3	fail	fail
$\bar{x}_1 = -1$	q_o	-	-	-	12.5	13.1	13.1	12.2	10.8	-	-
$\bar{x}_2 = 4$	x_1	-	-	-	3.98	3.8	**3.41**	2.87	2.28	-	-
$\kappa = 30$	x_2	-	-	-	0.28	**0.87**	1.14	1.8	1.98	-	-
k=2	No. of iterat.	fail	3	3	3	3	3	3	3	3	3
$\bar{x}_1 = 3.41$	q_o	-	13.2	13.2	13.2	13.2	13.2	13.2	13.2	13.2	13.1
$\bar{x}_2 = 0.87$	x_1	-	3.78	3.76	3.73	**3.69**	**3.65**	**3.61**	3.56	3.51	3.46
$\kappa = 4$	x_2	-	0.91	0.96	1.01	**1.07**	**1.12**	**1.19**	1.25	1.31	1.37

Total length 8 iterations, compared
to 21 iterations without parallelization

Similar experiments with $P = 5$ gave total length of 12 iterations compared to 21. The speedup of decreasing the total length of computations to approximately one-half or one-third when using 5 to 10 processors is not very high. On the other hand, note that this parallelization principle produces many results that are rather close to the solution $\hat{\boldsymbol{x}}$ of the original problem; this property is advantageous for badly conditioned problems (to which belongs also the example of a discretized circle), since it increases the robustness of the algorithm. We must be prepared to pay in computational effort for this property; if we apply a rule of thumb that we consider this property equally desirable as shortening the total length of computations, we should be satisfied with a speedup close to $\sqrt{P}$.

Note that the results could be improved because $n+1 = 3$ (instead of 1) iterations are used in DIDAS-L to cut through the interior of X_o and the initial feasibility phase might be shortened when using the properties of the augmented formulation. However, such a simple example with $n = 2$ might be misleading: it is more difficult to develop a concept for parallelization if $n > 2$.

On the other hand, the results of this simple example justify several conclusions concerning possible details of an augmented simplex algorithm for larger n:

- An initial phase of finding a feasible $\bar{\boldsymbol{x}}$ should be added; since $\bar{\boldsymbol{x}}$ might be any internal point of X_o , methods different than in typical simplex feasibility phases should be exploited;

- A long final single-criteria phase might be shortened by restarting the algorithm, e.g. (when using the lower bound strategy) from a new internal $\bar{\boldsymbol{x}}$ and cutting again through the interior of X_o;

- rules for adjusting starting points, e.g. $\bar{\boldsymbol{x}}$, and for choosing $\boldsymbol{d}$ in such *repetitive augmented simplex algorithm* and in its various possible parallel versions should be devised.

4 Algorithmic aspects.

4.1 Initial feasibility phase based on subdifferentiable optimization.

It is well known that, instead of solving problems (3) or (8a,b), an external exact penalty function (piece-wise linear penalty terms added to the original objective function; the latter should have reverse sign to keep with the tradition of minimizing penalties) could be minimized within simple bounds $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$. Since such a function is subdifferentiable, an algorithm of minimizing it would consist of two interchanged operations: solving a quadratic programming problem that determines the lowest Euclidean norm element of the subdifferential and then finding a directional minimum of the function. However, such a method would usually require much more computations than the simplex method: the number of constraints in the quadratic problems, low in the beginning iterations, would gradually increase to the number of active constraints at the final solution, and the computational effort to solve each of them might be higher than when solving the problem by simplex method.

Under certain conditions, however, *a subdifferentiable algorithm might be well competitive with the initial phase of a simplex algorithm* — as long as the number of constraints in related quadratic programming problems will be low.

Consider thus the following exact penalty term:

$$\Phi(\boldsymbol{x}) = \max_{1 \leq i \leq m} (\max(0, b_i - \boldsymbol{a}^{(i)T}\boldsymbol{x}) + \max(0, \boldsymbol{a}^{(i)T}\boldsymbol{x} - b_i - r_i)) \tag{12a}$$

Finding a feasible point means finding a minimum $\bar{\boldsymbol{x}}$ of $\Phi(\boldsymbol{x})$ w.r.t. $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$ such that $\Phi(\bar{\boldsymbol{x}})=0$. The subdifferential of Φ has the form:

$$\partial\Phi(\boldsymbol{x}) = \{\boldsymbol{d} \in \boldsymbol{R}^n : \boldsymbol{d} = -\sum_{i \in I_a^l(\boldsymbol{x})} \lambda_i \boldsymbol{a}^{(i)} + \sum_{i \in I_a^u(\boldsymbol{x})} \mu_i \boldsymbol{a}^{(i)}\} \tag{12b}$$

where:

$$I_a^l(\boldsymbol{x}) = \{i = 1, \ldots m : b_i - \boldsymbol{a}^{(i)T}\boldsymbol{x} = \Phi(\boldsymbol{x})\} \tag{12c}$$

$$I_a^u(\boldsymbol{x}) = \{i = 1, \ldots m : \boldsymbol{a}^{(i)T}\boldsymbol{x} - b_i - r_i = \Phi(\boldsymbol{x})\} \tag{12d}$$

$$\sum_{i \in I_a^l(\boldsymbol{x})} \lambda_i + \sum_{i \in I_a^u(\boldsymbol{x})} \mu_i = 1,\ \lambda_i, \mu_i \geq 0\ \forall i = 1, \ldots m,\ \lambda_i = 0\ \forall i \notin I_a^l(\boldsymbol{x}),\ \mu_i = 0\ \forall i \notin I_a^u(\boldsymbol{x}) \tag{12e}$$

In a typical subdifferentiable algorithm, the following quadratic programming problem would be solved:

$$\hat{\boldsymbol{d}} = \underset{\boldsymbol{d} \in \partial\Phi(\boldsymbol{x})}{\operatorname{argmin}} \|\boldsymbol{d}\|^2 \tag{12f}$$

in order to check whether a minimum is found (if $\hat{\boldsymbol{d}} = 0$) or to use $\boldsymbol{d} = -\hat{\boldsymbol{d}}$ in a subsequent directional search. However, there are simpler tests (e.g. $\Phi(\boldsymbol{x}) = 0$) to check that a minimum is found in this case and any $\boldsymbol{d} \in -\partial\Phi(\boldsymbol{x})$ can be used for subsequent directional search. Thus, we shall approximate the solution of (12f) and apply a simple but effective directional search. The resulting *subdifferential feasibility phase algorithm* is as follows.

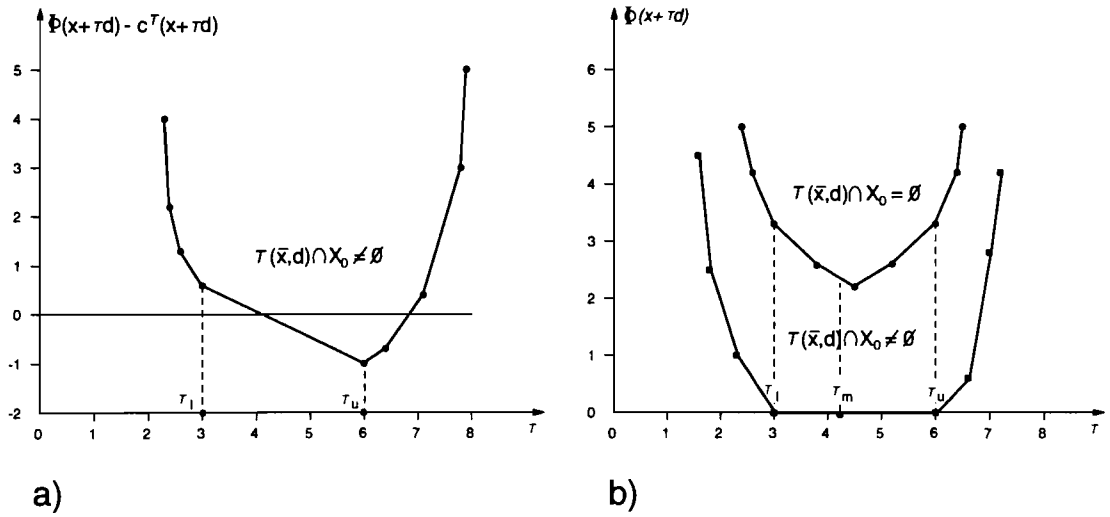

Figure 2: An illustration of the subdifferential feasibility algorithm.
a) full penalty approach,
b) penalty term for infeasibilities.

1^o Starting with given $\bar{\boldsymbol{x}}, \boldsymbol{d}$ and assuming $\boldsymbol{x} = \bar{\boldsymbol{x}} + \tau\boldsymbol{d}$, determine (similarly as when choosing a variable that leaves the basis in a simplex iteration, see Appendix 1) the values τ_u, τ_l of the coefficient τ that result in satisfying either all constraints which increase along $\boldsymbol{d}$ or all constraints which decrease along $\boldsymbol{d}$;

2^o Set τ_m as the arithmetic mean of τ_u, τ_l and shift $\bar{\boldsymbol{x}} := \bar{\boldsymbol{x}} + \tau_m\boldsymbol{d}$. If $\tau_l < \tau_u$ (which indicates that $T(\bar{\boldsymbol{x}}, d) \cap \operatorname{Int} X_o \neq \emptyset$ and $\Phi(\bar{\boldsymbol{x}}) = 0$) - stop.

3^o Solve (12f) for an approximate subdifferential while taking into account at most two elements $-\boldsymbol{a}^{(i)}$ or $\boldsymbol{a}^{(i)}$ that span (12b). Due to this approximation, one iteration of this algorithm requires a computational effort not exceeding the effort of one simplex iteration. Set $\boldsymbol{d} \cong -\hat{\boldsymbol{d}}$, return to 1^o.

The algorithm is presented in more detail in Appendix 1; the main features of the algorithm are illustrated in Fig. 2.

The subdifferentiable feasibility algorithm is not always effective. If a (uniform) measure of the set X_o is much smaller than the measure of the set $B_o = \{\boldsymbol{x} \in \boldsymbol{R}^n : \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}\}$ — see Fig. 3 — then a more standard simplex feasibility phase might be more effective, with a computational effort depending on the number of violated constraints, not on

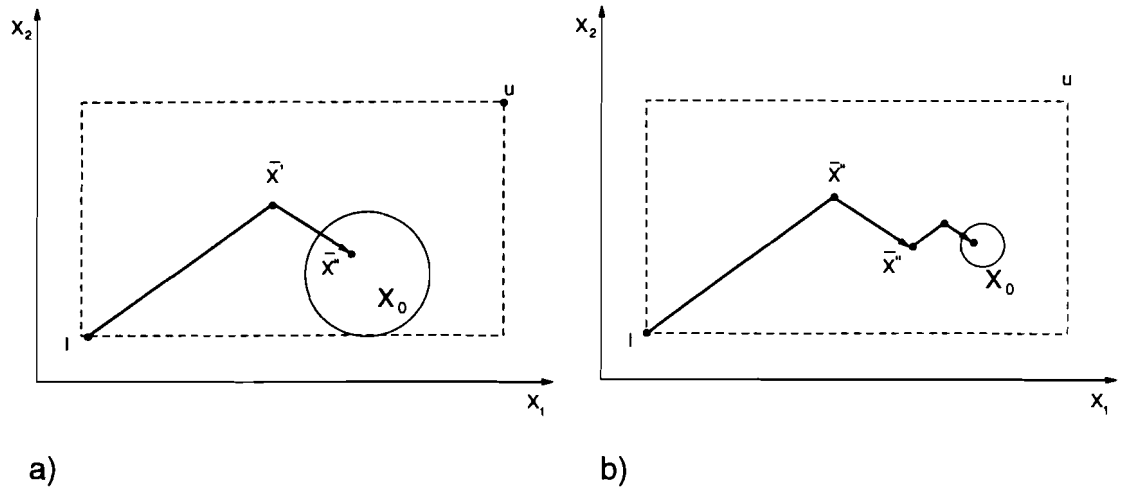

Figure 3: The impact of the relative measure of the set X_o:
a) relatively large measure,
b) relatively small measure.

the measures of these sets. Thus, the subdifferentiable feasibility algorithm might be used optionally in the initial phase, but it can be always used e.g. in further iterations of a repetitive augmented simplex algorithm, when an adjusted $\bar{\boldsymbol{x}}$ might happen to be infeasible.

A rough implementation of the subdifferentiable feasibility algorithm for the example of discretized circle was prepared and its results compared with the number of iterations needed for the initial phase of a standard simplex algorithm. The results are given in Table 3. Although this comparison is not quite fair - the discretized circle exaggerates the difficulties of a simplex algorithm - these results are rather striking.

Table 2. Comparison of standard with subdifferentiable feasibility phase

Bounds	l_1	0	-3	-6	-77	-77	-77
	l_2	-2	-5	-8	-77	-177	-977
	u_1	5	5	5	23	23	23
	u_2	3	3	3	3	23	23
Iterations standard f. ph.		27	42	45	45	45	45
Iterations subdiff. f. ph.		1	1	2	3	4	6

4.2 A repetitive augmented simplex algorithm.

When starting an augmented simplex algorithm with a relative lower bound strategy, a point $\bar{\boldsymbol{x}}$ situated in the middle of $B_o = \{\boldsymbol{x} \in \boldsymbol{R}^n : \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}\}$ might be selected,

e.g. $\bar{\boldsymbol{x}} = 0.5(\boldsymbol{u} + \boldsymbol{l})$. The starting direction might be either $\boldsymbol{d} = \boldsymbol{c}$ or $\boldsymbol{d} = \boldsymbol{u} - \boldsymbol{l}$. After correcting $\bar{\boldsymbol{x}}$ by the subdifferential feasibility algorithm, we might use yet different $\boldsymbol{d} = \bar{\bar{\boldsymbol{x}}} - \bar{\boldsymbol{x}}$, where either $\bar{\bar{\boldsymbol{x}}} = \boldsymbol{u}$ or both $\bar{\boldsymbol{x}}$ and $\bar{\bar{\boldsymbol{x}}}$ are suitably modified after restarting the augmented simplex algorithm.

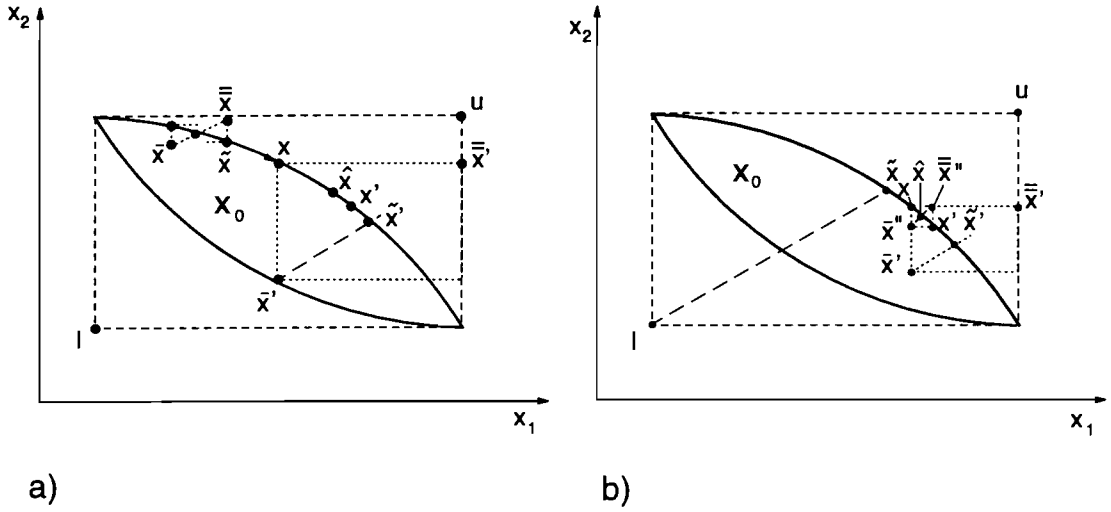

Figure 4: An illustration of a repetitive augmented simplex algorithm.

Recall that $\bar{\boldsymbol{x}}$ and $\bar{\bar{\boldsymbol{x}}}$ denote relative lower and upper bounds for the solution $\hat{\boldsymbol{x}}$, while we know that $\hat{\boldsymbol{x}} \in \hat{X}_o$. We shall use a heuristic supposition to estimate new $\bar{\boldsymbol{x}}$ and $\bar{\bar{\boldsymbol{x}}}$. Suppose that a feasible $\bar{\boldsymbol{x}}$ is found and the multiple-objective phase is completed at a point $\tilde{\boldsymbol{x}} \in \hat{X}_o$. The single-objective phase starts from $\tilde{\boldsymbol{x}}$, but only a relatively small number of simplex iterations $\kappa_o \geq 2$ is performed in this phase which results in another point $\boldsymbol{x} \in \hat{X}_o$. During these iterations, the monotonicity of changes of each variable might be tested. The supposition is that *if the changes of a variable $\boldsymbol{x}_j$ are monotone, they will continue to be monotone until its optimal value $\hat{x}_j$ is reached.* In this supposition, the concept of monotonicity might be variously interpreted: either as *strict monotonicity*, that each change of a variable must have the same sign, or as *relaxed monotonicity*, that the total change during κ_o iterations is much larger than the sum of changes with opposite sign.

The monotonicity supposition might be not true (imagine the surface of a sphere approximated by small triangles); therefore, we must check whether $\boldsymbol{x}$ is contained within the relative bounds $\bar{\boldsymbol{x}}$, $\bar{\bar{\boldsymbol{x}}}$ after κ_o simplex iterations of the single- objective phase and relax these bounds if needed. Even if the current solution $\boldsymbol{x}$ violates the relative bounds, we should use the available information to establish new relative bounds.

For example, if we accidentally tightened relative bounds too much, as in Fig. 3a, and obtain some $x_j < \bar{x}_j$, this means that x_j is essentially decreasing. We should set then $\bar{\bar{x}}_j := x_j$ but, at the same time, relax $\bar{x}_j$ rather strongly; we can take then $\bar{x}_j$ "all the way

through X_o " — see Appendix 2 for details. The same applies to all other variables for which new estimates $\bar{\bar{\boldsymbol{x}}}$ are established by monotonicity supposition.

On the other hand, if the repetitive augmented simplex proceeds normally without violating relative bounds, it might give best results after new estimates for all $\bar{x}_j$ are obtained by monotonicity supposition, see Fig. 3b.

The heuristic rules discussed above are summarized in an outline of *a repetitive augmented simplex algorithm for sequential computations* presented in Appendix 2. The algorithm is constructed in such a way that it is convergent in a finite number of iterations.

4.3 A parallel version of the augmented simplex algorithm.

For a parallel version of the augmented simplex, heuristic rules of choosing directions $\boldsymbol{d}$ are needed. If the relative lower bound strategy is used, this choice should result in possibly best estimates of the relative bounds $\bar{\boldsymbol{x}}$, while the bounds $\bar{\bar{\boldsymbol{x}}}$ play an auxiliary role.

It is known in the theory of multi-objective optimization that finding a precise lower bound (called sometimes *a nadir point*) for a Pareto set is a rather difficult computational problem. However, experience in estimating approximate bounds (e.g. in DIDAS-like systems) suggests a heuristic rule: to estimate lower bounds, reference points should be displaced along vectors $\boldsymbol{m}^{(j)} = (1 \ldots 1 \ 0_{(j)} \ 1 \ldots 1)^T$, while displacements along versors $\boldsymbol{e}^{(j)} = (0 \ldots 0 \ 1_{(j)} \ 0 \ldots 0)^T$ should be used in the case of upper bounds. Clearly, $\{\boldsymbol{m}^{(j)}\}$ differs from $\{\boldsymbol{e}^{(j)}\}$ if $n \geq 3$.

Therefore, at least $2n+2$ *essential directions* $\boldsymbol{d}^{(\psi)}$, $\psi = 1, \ldots 2n+2$, including $\boldsymbol{d}^{(1)} = c$; $\boldsymbol{d}^{(2)} = \bar{\bar{\boldsymbol{x}}} - \bar{\boldsymbol{x}}$ as well as all vectors $\boldsymbol{m}^{(j)}$, $\boldsymbol{e}^{(j)}$ should be used in a parallel version of the augmented simplex algorithm (see Appendix 3 for more details).

However, in an iteration of the parallel and repetitive augmented simplex algorithm, these directions generate a cluster of points $\boldsymbol{x}^{(\psi)}$ (each taken at the end of κ_o iterations of the single-objective phase). These points can be appropriately re-ordered according to the values of objective function. A convex combination of the best n of these results (with the highest values of the objective function) can be especially useful for generating an additional essential direction; this is a heuristic conclusion from Mazur theorem on the convergence of convex combinations. Thus (in next iterations) a preferred essential direction $\boldsymbol{d}^{(0)}$ corresponding to such convex combination of best $\boldsymbol{x}^{(\psi)}$ should be added.

Moreover, *auxiliary directions* $\boldsymbol{d}^{(\psi)}, \psi = 2n+3, \ldots \psi_u$ might be useful, if they concentrate rather closely around $\boldsymbol{d}^{(0)}$, $\boldsymbol{d}^{(1)}$, $\boldsymbol{d}^{(2)}$. Such auxiliary directions might be generated as convex combinations of $\boldsymbol{d}^{(0)}$, $\boldsymbol{d}^{(1)}$, $\boldsymbol{d}^{(2)}$ with other essential directions.

Thus, if P processors are available for parallel computations, we should choose $\psi_u > 3n+2$ such that $\chi_o = (\psi_u + 1)/P$ is an integer, indicating how many times a processor will be used in one large iteration — and then generate enough auxiliary directions to use fully the available processors. The times of executing an assigned number of large iterations of the repetitive augmented simplex algorithm might vary between the processors. The assignment of tasks for processors should be such that parameterized problems with more essential directions should be run first and the less preferred problems with auxiliary directions and larger ψ should be run last on all processors. We can stop then all processors at the same time — say, when at least $\psi_m \geq 2n+3$ results $\boldsymbol{x}^{\psi}$ are available — without the danger of loosing much results for more preferred directions.

The data obtained from all processors must be appropriately used for establishing new relative bounds $\bar{\boldsymbol{x}}$ and $\bar{\bar{\boldsymbol{x}}}$, see Appendix 3 for details. All this results in an outline of a parallel version of the augmented simplex algorithm, presented also in Appendix 3. While most discussions of this paper, for exposition purposes, concentrate on the relative lower bound strategy, it should be stressed again that a relative upper bound strategy — choosing $\bar{\bar{\boldsymbol{x}}}$ as a starting point — is also possible.

A rough implementation of the repetitive augmented simplex algorithm and a simulation of its parallel versions were tested on the example of discretized circle (with simplifications related to $n = 2$) and predictably gave better results than those presented in earlier sections; the results for relative upper bound strategy were slightly better than those for lower bound strategy. However, such results for $n = 2$ might be misleading; a full implementation and much more extensive testing are intended.

5 Conclusions.

A concept of a modification of the standard simplex algorithm was presented, based on an imbedding into a multiple-objective family and exploiting some ideas of subdifferentiable optimization and of interior point methods. The resulting repetitive augmented simplex algorithm might be competitive with interior point methods in such cases when a standard simplex algorithm would spend much time on processing consecutive vertices.

Moreover, some of the properties of this algorithm might be useful in various modifications of linear programming algorithms, such as alternative initial feasibility phases or crash updates of the basis. The property of starting with any internal point and then finding vertices of the simplex might provide an alternative way of recovering a vertex solution for interior point methods, see e.g. Mehrotra (1991). However, *the main advantage of the proposed algorithm is the possibility of its coarse-grained parallelization* which might indicate a way of using future computers of scalably parallel structure.

The proposed algorithms must be yet fully implemented and extensively tested. However, an early publication of these conceptual results was deemed important, since the concept of a parametric imbedding of optimization algorithms for their coarse-grained parallelization might be also used in any other classes of problems in nonlinear and discrete optimization.

References

[1] Arbel, A. (1992) Using Sequential Generation of Anchoring Points in an Interior Multiobjective Primal-Dual Linear Programming Algorithm. Working Paper of Tel-Aviv University.

[2] Bertsekas D.P. and J.N. Tsitsiklis (1989) *Parallel and Distributed Computation.* Prentice Hall, Englewood Cliffs.

[3] Gal, T. (1977) A general method of determining the set of all efficient solutions to a linear vectormaximum problem. *European Journal of Operational Research,* Vol. 1, No. 5, pp. 307-322.

[4] Gabasov, R. and F.M. Kiryllova (1977) *Linear Programming Methods* (in Russian). Isdatelstvo BGU, Minsk.

[5] Gondzio, J. and M. Makowski (1993) Solving A Class of LP Problems with a Primal-Dual Logarithmic Barrier Method. Working Paper of the International Institute for Applied Systems Analysis, WP-93-02, Laxenburg, Austria.

[6] Karmarkar, N.K. (1984) A new polynomial time algorithm for linear programming. *Combinatorica,* Vol. 4, pp. 373-395.

[7] Lewandowski, A. and A.P. Wierzbicki, eds. (1989) *Aspiration Based Decision Support Systems.* Lecture Notes in Economics and Mathematical Systems, Vol. 331, Springer-Verlag, Berlin-Heidelberg.

[8] Makowski, M. and J.S. Sosnowski (1989) Mathematical programming package HYBRID. In A. Lewandowski and A.P. Wierzbicki, eds., op.cit.

[9] Mehrotra, S. (1991) On finding a vertex solution using interior point methods. *Linear Algebra and Its Applications,* Vol. 152, pp. 233-253.

[10] Murtagh, B.A. and M.A. Saunders (1977) MINOS - a Large Scale Nonlinear Programming System for Problems with Linear Constraints. User Guide. Technical Report of Stanford Optimization Laboratory.

[11] Nakayama, H. and Y. Sawaragi (1983) Satisficing trade-off method for multiobjective programming. In M. Grauer and A.P. Wierzbicki, eds.: *Interactive Decision Analysis.* Lecture Notes in Economics and Mathematical Systems, Vol. 229, Springer-Verlag, Berlin-Heidelberg.

[12] Nogi, T. (1986) Parallel computation. In: Patterns and Waves - Qualitative Analysis of Nonlinear Differential Equations, *Studies in Mathematics and Applications* Vol. 18 pp. 279-318.

[13] Rogowski T., J. Sobczyk and A.P. Wierzbicki (1989) IAC-DIDAS-L - A dynamic interactive decision analysis and support system for multicriteria analysis of linear and dynamic linear models on professional microcomputers. In A. Lewandowski and A.P. Wierzbicki, eds., op.cit.

[14] Ruszczyński, A. (1992) An augmented Lagrangian decomposition method for block diagonal linear programming problems. *Operations Research Letters,* Vol. 8, pp. 287-294.

[15] Seo, F. and M. Sakawa (1988) *Multiple Criteria Decision Analysis in Regional Planning: Concepts, Methods and Applications.* D. Reidel, Dordrecht.

[16] Steuer, R.E. (1986) *Multiple Criteria Optimization: Theory, Computation, and Application.* J.Wiley & Sons, New York.

[17] Steuer, R.E. and E.V. Choo (1983) An interactive weighted Chebyshev procedure for multiple objective programming. *Mathematical Programming* Vol. 26, pp. 326-344.

[18] Wierzbicki, A.P. (1980) The use of reference objectives in multiobjective optimization. In G. Fandel and T. Gal, eds., *Multiple Criteria Decision Making, Theory and Applications.* Springer-Verlag, Heidelberg.

[19] Wierzbicki, A.P. (1986) On the completeness and constructiveness of parametric characterizations to vector optimization problems. *OR-Spektrum,* Vol. 8, pp. 73-87.

[20] Wierzbicki, A.P. (1992a) Multiple Criteria Games - Theory and Applications. Working Paper of the International Institute for Applied Systems Analysis, WP-92-079, Laxenburg, Austria.

[21] Wierzbicki, A.P. (1992b) Multi-Objective Modeling and Simulation for Decision Support. Working Paper of the International Institute for Applied Systems Analysis, WP-92-080, Laxenburg, Austria.

[22] Yu, P.L and M. Zeleny (1975) The set of all non-dominated solutions in linear cases and a multicriteria simplex method. *Journal of Mathematical Analysis and Applications,* Vol. 49, pp. 430-468.

A Appendices.

A.1 A subdifferential feasibility algorithm.

The algorithm presented in the main text is described here in somewhat more detail:

1^o Starting with given $\bar{\boldsymbol{x}}$, $\boldsymbol{d}$ and assuming $\boldsymbol{x} = \bar{\boldsymbol{x}} + \tau \boldsymbol{d}$, determine the values of the coefficient τ that result in satisfying either all constraints which increase along $\boldsymbol{d}$ or all constraints which decrease along $\boldsymbol{d}$:

$$\tau_{um} = \tau_{um}(\bar{\boldsymbol{x}}, \boldsymbol{d}) = \min_{j=1,\dots n} (u_j - \bar{x}_j)/d_j \tag{13a}$$

$$\tau_{lm} = \tau_{lm}(\bar{\boldsymbol{x}}, \boldsymbol{d}) = \max_{j=1,\dots n} (l_j - \bar{x}_j)/d_j \tag{13b}$$

$$\tau_u = \tau_u(\bar{\boldsymbol{x}}, \boldsymbol{d}) = \min(\tau_{um}, \min_{i \in I^l} \frac{b_i - \boldsymbol{a}^{(i)T}\bar{\boldsymbol{x}}}{\boldsymbol{a}^{(i)T}\boldsymbol{d}}, \min_{i \in I^u} \frac{b_i + r_i - \boldsymbol{a}^{(i)T}\bar{\boldsymbol{x}}}{\boldsymbol{a}^{(i)T}\boldsymbol{d}}) \tag{13c}$$

$$\tau_l = \tau_l(\bar{\boldsymbol{x}}, d) = \max(\tau_{lm}, \max_{i \in I^u} \frac{b_i - \boldsymbol{a}^{(i)T}\bar{\boldsymbol{x}}}{\boldsymbol{a}^{(i)T}\boldsymbol{d}}, \max_{i \in I^l} \frac{b_i + r_i - \boldsymbol{a}^{(i)T}\bar{\boldsymbol{x}}}{\boldsymbol{a}^{(i)T}\boldsymbol{d}}) \tag{13d}$$

where:

$$I^u = \{i = 1, \dots m : \boldsymbol{a}^{(i)T}\boldsymbol{d} > 0\},\ I^l = \{i = 1, \dots m : \boldsymbol{a}^{(i)T}\boldsymbol{d} < 0\}. \tag{13e}$$

Note that $\tau_u \leq \tau_l$ indicates that $T(\bar{\boldsymbol{x}}, \boldsymbol{d}) \cap \operatorname{Int} X_o = \emptyset$, but in this case a middle point between these points might be taken as an approximate directional minimum of $\Phi(\boldsymbol{x} + \tau \boldsymbol{d})$.

2^o Set:

$$\bar{\boldsymbol{x}} := \bar{\boldsymbol{x}} + \tau_m \boldsymbol{d} \tag{13f}$$

where:

$$\tau_m = \tau_m(\bar{\boldsymbol{x}}, \boldsymbol{d}) = 0.5(\tau_l + \tau_u) \tag{13g}$$

If $\tau_l < \tau_u$ — which indicates that $T(\bar{\boldsymbol{x}}, \boldsymbol{d}) \cap \operatorname{Int} X_o \neq \emptyset$ and $\Phi(\bar{\boldsymbol{x}}) = 0$) - stop. Note that the algorithm is not stopped if $\tau_l = \tau_u$, because this would imply $\bar{\boldsymbol{x}}$ on the boundary of X_o and we can use subgradient information in such a case to find a point in the interior of X_o .

3^o Determine $I_a^l(\bar{\boldsymbol{x}}), I_a^u(\bar{\boldsymbol{x}})$ as in (12c,d). Set

$$\theta = |I_a^l(\bar{\boldsymbol{x}}) \cup I_a^u(\bar{\boldsymbol{x}})|, \ \{\boldsymbol{d}^{(j)}\}_{j=1}^{\theta} = \{-\boldsymbol{a}^{(i)}\}_{i \in I_a^l(\bar{\boldsymbol{x}})} \cup \{\boldsymbol{a}^{(i)}\}_{i \in I_a^u(\bar{\boldsymbol{x}})}. \tag{13h}$$

If $\theta = 1$, set $\boldsymbol{d} := -\boldsymbol{d}^{(1)}$, go to 1^o. Determine Euclidean norms $\|\boldsymbol{d}^{(j)}\|$, reorder $\boldsymbol{d}^{(j)}$ in such a way that $\|\boldsymbol{d}^{(1)}\| \leq \|\boldsymbol{d}^{(2)}\| \leq \ldots \leq \|\boldsymbol{d}^{(\theta)}\|$. Determine:

$$\lambda = \frac{\|\boldsymbol{d}^{(2)}\|^2 - \boldsymbol{d}^{(2)T}\boldsymbol{d}^{(1)}}{\|\boldsymbol{d}^{(2)} - \boldsymbol{d}^{(1)}\|^2} \tag{13i}$$

If $\lambda < 0$, set $\lambda := 0$. If $\lambda > 1$, set $\lambda := 1$. Set:

$$\boldsymbol{d} := -\lambda \boldsymbol{d}^{(1)} - (1-\lambda)\boldsymbol{d}^{(2)}; \ \text{goto} \ 1^o. \tag{13j}$$

Step 3 solves approximately problem (12f). The most probable numbers of elements $\boldsymbol{d}^{(j)}$, $j = 1, \ldots \theta$, spanning the subdifferential $\partial\Phi(\boldsymbol{x})$ in this algorithm are $\theta = 1$ or 2. If $\theta = 1$, problem (12f) is trivial. If $\theta = 2$, this problem is solved precisely: an element $\boldsymbol{d} = \lambda \boldsymbol{d}^{(1)} + (1-\lambda)\boldsymbol{d}^{(2)}$ of lowest norm, if λ would not be constrained, would be orthogonal to $\boldsymbol{d}^{(1)} - \boldsymbol{d}^{(2)}$ which implies (13i); because $\lambda \in [0; 1]$, it is sufficient to project its value on this interval. If $\theta > 2$, the solution of (12f) is approximated by taking into account only two $\boldsymbol{d}^{(1)}$ and $\boldsymbol{d}^{(2)}$ that have the lowest norms. Due to this approximation, one iteration of this algorithm requires a computational effort not exceeding the effort of one simplex iteration.

Only the basic elements of the algorithm are presented above; they should be supplemented with such details as counting the number of iterations and limiting it to some reasonable value ϑ_o — e.g. to a fraction of $n+m$ — or projecting the directions on bounds in the case that a component x_j is equal to l_j or u_j .

A.2 An outline of a repetitive simplex algorithm for sequential computations.

Two cases are distinguished in the algorithm: the monotonicity test is either satisfied or violated.

In the latter case, the variables that violate the relative bounds are distinguished from others. If the relative bounds are tightened too much, and some $x_j < \bar{x}_j$ is obtained, it is assumed that x_j is essentially decreasing. Therefore, $\bar{\bar{x}}_j := x_j$ is set and, at the same time, $\bar{x}_j$ is rather strongly relaxed: it might be taken "all the way through X_o " using the operation $\tau_u(\boldsymbol{x}, -\boldsymbol{e}^{(j)})$ defined as in (13c) with $\boldsymbol{e}^{(j)} = (0 \ldots 0 \ 1_{(j)} \ 0 \ldots 0)^T$. The same applies in this case to all other variables which change monotone and for which new estimates $\bar{\bar{\boldsymbol{x}}}$ are established.

If monotonicity tests justify a new estimate $\bar{x}_j$ after a violation of relative bounds, $\bar{\bar{x}}_j$ should be strongly relaxed by taking it to u_j. For variables that do not change monotone,

original $\bar{x}_j$ and $\bar{\bar{x}}_j$ might be reset. Additionally, since a violation of relative bounds might indicate a problem difficult to solve, κ_o should be increased — say, by setting $\kappa_o := 2\kappa_o$.

In the former case, when the repetitive augmented simplex proceeds normally without violating relative bounds, new estimates for all $\bar{x}_j$ are obtained by monotonicity supposition. As long as only $\bar{\bar{x}}_j := x_j$ is set and $\bar{x}_j$ was not modified in such a way yet, $\bar{x}_j$ might be adjusted by taking it "half-way through X_o " while using the operation $\tau_m(\boldsymbol{x}, -\boldsymbol{e}^{(j)})$ defined as in (13g).

The heuristic rules discussed above can be summarized in the following outline of *a repetitive augmented simplex algorithm for sequential computations*:

1^o Choose an option of starting with a standard or a subdifferentiable feasibility phase, of interpreting the monotonicity supposition and of using $\boldsymbol{d} = \boldsymbol{c}$ or $\boldsymbol{d} = \bar{\bar{\boldsymbol{x}}} - \bar{\boldsymbol{x}}$; select parameters κ_o and ϑ_o . Set $k := 1$, $\bar{\boldsymbol{x}} := l$ and use the subdifferentiable feasibility phase algorithm for at most ϑo of its iterations. If $\boldsymbol{x} \in \operatorname{Int} X_o$ is not found, apply a standard simplex feasibility phase starting from last $\bar{\boldsymbol{x}}$.

2^o Apply the multiple-objective phase of the augmented simplex algorithm until $\tilde{\boldsymbol{x}} \in \hat{X}_o$ is found; then perform κ_o iterations of the single-objective phase to obtain $\boldsymbol{x}$ and set:

$$\begin{array}{l} \zeta_j := 1 \text{ if } x_j \text{ monotone increases,} \\ \zeta_j := -1 \text{ if it monotone decreases,} \\ \zeta_j := 0 \text{ otherwise} \end{array} \tag{14a}$$

initialize updating of $\bar{\boldsymbol{x}}$ and $\bar{\bar{\boldsymbol{x}}}$:

$$\text{if } k = 1, \text{ set } \xi_j := 0 \; \forall j = 1, \ldots n, \tag{14b}$$

else test whether a violation of relative bounds occurred:

$$\text{if } x_j \notin [\bar{x}_j;\ \bar{\bar{x}}_j] \text{ for any } j = 1, \ldots n, \text{ go to } 4^o. \tag{14c}$$

3^o Perform *a normal update of relative bounds*; for all $j = 1, \ldots n$:

$$\text{if } \zeta_j > 0, \text{ set } \bar{x}_j := x_j, \; \xi_j := 1, \tag{14d}$$

$$\text{if } \zeta_j < 0, \text{ set } \bar{\bar{x}}_j := x_j \text{ and if } \xi_j = 0, \text{ set } \bar{x}_j := x_j - \tau_m(\boldsymbol{x}, -\boldsymbol{e}_{(j)}) \tag{14e}$$

Set $\boldsymbol{d} := \boldsymbol{c}$ or $\boldsymbol{d} := \bar{\bar{\boldsymbol{x}}} - \bar{\boldsymbol{x}}, k := k + 1$, go to 2^o .

4^o Perform *an update of violated relative bounds*; for all $j = 1, \ldots n$ set $\xi_j := 0$ and:

$$\text{if } x_j > \bar{\bar{x}}_j \text{ or if } \zeta_j > 0, \text{ set } \bar{x}_j := x_j, \xi_j := 1 \text{ and } \bar{\bar{x}}_j := u_j \tag{14f}$$

$$\text{if } x_j < \bar{x}_j \text{ or if } \zeta_j < 0, \text{ set } \bar{\bar{x}}_j := x_j, \; \bar{x}_j := x_j - \tau_u(\boldsymbol{x}, -\boldsymbol{e}^{(j)}) \tag{14g}$$

else (if not (14f, g) and) if $\zeta_j = 0$, set $\bar{\bar{x}}_j := u_j$, $\bar{x}_j := x_j - \tau_u(\boldsymbol{x}, -\boldsymbol{e}^{(j)})$ (14h)

Set $\boldsymbol{d} := \boldsymbol{c}$ or $\boldsymbol{d} := \bar{\bar{\boldsymbol{x}}} - \bar{\boldsymbol{x}}$, $k := k + 1$, test whether $\bar{\boldsymbol{x}} \in X_o$ by performing operations $(13a, \ldots e)$ of the subdifferential feasibility phase; if not, set $\bar{\boldsymbol{x}} := \bar{\boldsymbol{x}} + \tau_l(\bar{\boldsymbol{x}}, \boldsymbol{d})\boldsymbol{d}$. Set $\kappa := 2\kappa$, go to 2^o.

Various details can be further specified for this outline, e.g. if a standard simplex feasibility phase is used, it might start either from $\boldsymbol{l}$ or from $\boldsymbol{u}$; this phase should be followed by the operations $(13a, \ldots f)$ of the subdifferentiable feasibility phase in order to obtain an $\bar{\boldsymbol{x}} \in \operatorname{Int} X_o$. Various modifications of the rules of updating $\bar{\boldsymbol{x}}$, particularly after a violation of relative bounds, are also possible — for example, we could use more conservative updates based not on the current solution $\boldsymbol{x}$, but on the point $\tilde{\boldsymbol{x}}$ obtained after the multi-objective phase.

Note that the algorithm is convergent in a finite number of iterations (except, in practice, badly conditioned problems for which also a standard simplex might fail): normal updates of relative bounds result in an increased effectiveness of the algorithm when compared to a standard simplex algorithm, while updates of violated bounds result in increasing κ_o and eventually in using standard simplex algorithm longer in the single-objective phase until the optimal solution is found.

A.3 Details of parallel version of the augmented simplex algorithm.

The $2n + 2$ essential directions $d^{(\psi)}$, $\psi = 1, \ldots 2n + 2$ postulated for a parallel version of the augmented simplex algorithm are:

$$\begin{aligned} &\boldsymbol{d}^{(1)} = \boldsymbol{c};\ \boldsymbol{d}^{(2)} = \bar{\bar{\boldsymbol{x}}} - \bar{\boldsymbol{x}},\ \boldsymbol{d}^{(\psi)} = \boldsymbol{m}^{(\psi-2)} \text{ for } \psi = 3, \ldots n + 2, \\ &\boldsymbol{d}^{(\psi)} = \boldsymbol{e}^{(\psi-n-2)} \text{ for } \psi = n + 3, \ldots 2n + 2 \end{aligned} \tag{15a}$$

If such directions are used in a parallel implementation of the repetitive augmented simplex algorithm, various points $\boldsymbol{x}^{(\psi)}$ are obtained after a large iteration of the algorithm (each taken at the end of κ_o iterations of the single-objective phase). Since for all $\boldsymbol{x}^{(\psi)}$ also the values $q_o^{(\psi)} = \boldsymbol{c}^T\boldsymbol{x}^{(\psi)}$ are known, the results can be reordered according to the decreasing values of the objective function. They will be used appropriately for updating the relative bounds $\bar{\boldsymbol{x}}$, $\bar{\bar{\boldsymbol{x}}}$.

However, the effectiveness of the algorithm might increase considerably if a convex combination of the best n of these results (with the highest values of the objective function) were used in next large iterations for generating an additional essential direction:

$$\boldsymbol{d}^{(0)} = \sum_{\psi \in \Psi_n} \boldsymbol{x}^{\psi}/n - \bar{\boldsymbol{x}} \tag{15b}$$

where Ψ_n is the set of indices of n best $\boldsymbol{x}^{\psi}$. Moreover, auxiliary directions $\boldsymbol{d}^{(\psi)}$, $\psi = 2n + 3, \ldots \psi_u$ are useful, if they concentrate closely around $\boldsymbol{d}^{(0)}$, $\boldsymbol{d}^{(1)}$, $\boldsymbol{d}^{(2)}$. Such auxiliary directions might be generated as convex combinations of $\boldsymbol{d}^{(0)}$, $\boldsymbol{d}^{(1)}$, $\boldsymbol{d}^{(2)}$ with other essential directions:

$$\boldsymbol{d}^{(\psi)} = \lambda \boldsymbol{d}^{(i)} + (1-\lambda)\boldsymbol{d}^{(j)},\ i = 0,1,2,\ j = 3,\ldots 2n+2, \tag{15c}$$

where $\lambda \in (0;1)$ is either determined or a random variable; values λ close to 1 and $j = 3,\ldots n+2$ (corresponding to vectors $\boldsymbol{m}^{(j)}$ that are useful for estimating $\bar{\boldsymbol{x}}$) should be preferred. Additionally, other convex combinations (say, of all essential directions with random positive coefficients summing up to 1) might be used. It is advisable to use at least n auxiliary directions.

The directions are assigned to parallel processors as described in the main text. When stopping parallel computations after a large iteration, admissible data might come also from such processors that finished less than κ_o but at least 2 iterations in the single-objective phase (and thus the monotonicity indicators ζ_j^{ψ} are available for each $x_j^{(\psi)}$).

The data should be tested whether $\boldsymbol{x}^{(\psi)}$ do not exceed the relative bounds; however, since there is more data in the parallel case, *an update of violated relative bounds is needed only if, say, less than 3 best data is within the bounds.* If an update of violated relative bounds is not needed, the set of indices of admissible data Ψ is formed from all $\boldsymbol{x}^{\psi}$ that do not violate the relative bounds; set Ψ is ordered then according to decreasing values of $q_o^{(\psi)}$.

In a normal update of relative bounds, if the rules (14d,e) are used to establish new bounds and many admissible data $\boldsymbol{x}^{(\psi)}$ are available coming from various parts of the Pareto boundary of a polytope, various updates $\bar{x}_j^{(\psi)}$ and $\bar{\bar{x}}_j^{(\psi')}$ (usually for different ψ and ψ') are obtained and a rule of choosing between them is needed. A reasonable rule is that *priority is given to data corresponding to highest values of objective function* (to the lowest indices of j after reordering) and to updating $\bar{x}_j$. Thus, $\bar{x}_j^{(\psi)}$ and $\bar{\bar{x}}_j^{(\psi')}$ are chosen that correspond to lowest indices ψ and ψ', not such that give e.g. the most tight bounds. These updates might be contradictory in the sense that $\bar{\bar{x}}_j^{(\psi)}$ might be lower than $\bar{x}_j^{(\psi')}$, because the supposition of monotonicity might be not true. Following the adopted rule, the update $\bar{x}_j^{\psi}$ is chosen first and then a non-contradictory $\bar{\bar{\boldsymbol{x}}}_j^{(\psi')}$ is selected with the possibly lowest index ψ'.

These rules can be summarized in the following outline of a parallel version (based on lower bound strategy) of the augmented simplex algorithm:

1^o Choose an option of starting with a standard or a subdifferentiable feasibility phase, of interpreting the monotonicity supposition, parameters κ_o and ϑ_o and initial $\bar{\boldsymbol{x}}$, $\bar{\bar{\boldsymbol{x}}}$ as in the repetitive algorithm, set $k := 1$. Choose integers ψ_u, χ_o, ψ_m such that:

$$\chi_o P = \psi_u + 1 \geq 3n+3;\ \ \psi_m \geq 2n+3 \tag{16a}$$

Determine essential $\boldsymbol{d}^{(\psi)}$ as in (15a) for $j = 1,\ldots 2n+2$. Choose a rule of selecting auxiliary $\boldsymbol{d}^{(\psi)}$ as in (15c) for $\psi = 2n+3,\ldots \psi_u + 1$ and select them. Assign to each of P processors a first direction $\boldsymbol{d}^{(\psi)}$ starting with $\psi = 1,\ldots P$, then a second direction starting with lowest remaining ψ etc. until each processor has χ_o directions assigned.

2^o Run on each processor (if k=1) the feasibility phase and (for any k) the multiple-objective phase and κ_o iterations of the single- objective phase of the augmented simplex algorithm for each assigned direction $\boldsymbol{d}^{(\psi)}$, starting from a common $\bar{\boldsymbol{x}}$, to obtain $\boldsymbol{x}^{(\psi)}$ and the corresponding monotonicity indices $\zeta_j^{(\psi)}$, $j = 1,\ldots n$ (as in the repetitive algorithm)

together with the corresponding $q_o^{(\psi)} = \boldsymbol{c}^T \boldsymbol{x}^{(\psi)}$. Stop if at least ψ_m such data sets for various ψ are available (while accepting data from all processors that have monotonicity indicators $\zeta_j^{(\psi)}$ available).

Initialize updating of $\bar{\boldsymbol{x}}$ and $\bar{\bar{\boldsymbol{x}}}$: if $k = 1$, set $\xi_j := 0$ for all $j = 1, \ldots n$. Determine the set Ψ of admissible data:

$$\Psi = \{\psi : x_j^{(\psi)} \in [\bar{x}_j; \bar{\bar{x}}_j] \; \forall j = 1, \ldots n\} \tag{16b}$$

Reorder Ψ in such a way that $q_o^{(\psi)} \leq q_o^{(\psi+1)}$ for all $\psi = 1, \ldots |\Psi|$. If $|\Psi| < 3$, determine the smallest $\psi = \psi' \leq 3$ for which a violation of bounds $\bar{\boldsymbol{x}}$, $\bar{\bar{\boldsymbol{x}}}$ occurred, go to 4^o .

3^o Perform *a normal update of relative bounds*; for all $j = 1, \ldots n$ select subsequent ψ (as reordered) until:

$$\zeta_j^{(\psi)} > 0; \text{ if such } \psi \leq |\Psi| \text{ is found, set } \bar{x}_j := x_j^{(\psi)}, \; \xi_j := 1 \tag{16c}$$

Then select subsequent ψ until $\zeta_j^{(\psi)} < 0$ and, if $\xi_j = 1$, $x_j^{(\psi)} > \bar{x}_j$; if such $\psi \leq |\Psi|$ is found, set:

$$\bar{\bar{x}}_j := x_j^{(\psi)} \text{ and if } \xi_j = 0, \text{ set } \bar{x}_j := x_j^{(\psi)} - \tau_m(\boldsymbol{x}^{(\psi)}, -\boldsymbol{e}^{(j)}). \tag{16d}$$

Go to 5^o.

4^o Perform *an update of violated relative bounds* - for all $j = 1, \ldots n$ set $\xi_j := 0$ and:

$$\text{if } x_j^{(\psi')} > \bar{\bar{x}}_j, \text{ set } \bar{x}_j := x_j^{(\psi')}, \; \xi_j := 1 \text{ and } \bar{\bar{x}}_j := u_j \tag{16e}$$

otherwise select subsequent $\psi \in \Psi \cup \{\psi'\}$ and perform (16c),

$$\text{if } x_j^{(\psi')} < \bar{x}_j, \text{ set } \bar{\bar{x}}_j := x_j^{(\psi')}, \; \bar{x}_j := x_j^{(\psi')} - \tau_u(\boldsymbol{x}^{(\psi')}, -\boldsymbol{e}^{(j)}) \tag{16f}$$

or select subsequent $\psi \in \Psi \cup \{\psi'\}$ and perform (16d) while using the operation τ_u instead of τ_m,

$$\text{else (if not (16e,f) and) if } \zeta_j = 0, \text{ set } \bar{\bar{x}}_j := u_j, \; \bar{x}_j := x_j^{(1)} - \tau_u(\boldsymbol{x}^{(1)}, -\boldsymbol{e}^{(j)}) \tag{16g}$$

Set $\boldsymbol{d} = \boldsymbol{c}$ or $\boldsymbol{d} = \bar{\bar{\boldsymbol{x}}} - \bar{\boldsymbol{x}}$, $k := k + 1$, test whether $\bar{\boldsymbol{x}} \in X_o$ by performing operations $(13a, \ldots e)$ of the subdifferential feasibility phase; if not, set $\bar{\boldsymbol{x}} := \bar{\boldsymbol{x}} + \tau_l(\bar{\boldsymbol{x}}, \boldsymbol{d})\boldsymbol{d}$. Set $\kappa_o := 2\kappa_o$.

5^o Set $k := k + 1$, $\Psi_n := \Psi \cap \{1, \ldots n\}$, $n' := |\Psi_n|$ and determine $\boldsymbol{d}^{(0)}$:

$$\boldsymbol{d}^{(0)} = \sum_{\psi \in \Psi_n} \boldsymbol{x}^{(\psi)}/n' - \bar{\boldsymbol{x}} \tag{16i}$$

Update $\boldsymbol{d}^{(2)} = \bar{\bar{\boldsymbol{x}}} - \bar{\boldsymbol{x}}$, reassign $\boldsymbol{d}^{(\psi)}$ to each processor as in 1^o but starting with $\boldsymbol{d}^{(0)}$, go to 2^o.

Various details of this algorithm have to be modified for the case of upper bound strategy, when starting the searches from $\bar{\bar{\boldsymbol{x}}}$ instead of $\bar{\boldsymbol{x}}$.