Concave Programming under Simplest Linear Constraints

A. I. Rusakov

Rostov State University of Transport Communications, pl. Narodnogo opolcheniya 2,
Rostov-na-Donu, 344038 Russia

e-mail: rusakov@rost.ru

Received May 31, 2001; in final form, October 9, 2002

Abstract—A stable algorithm for a concave programming problem on a feasible polyhedron determined by bounds and a scalar equality condition is proposed. The algorithm is based on the branch and bound scheme, where the tested regions are cut off by successively reducing the size of a hyperparallelepiped that encloses the feasible region.

1. INTRODUCTION

There exist several approaches to numerical solution of linearly constrained concave programming problems. For various reasons, all of the corresponding methods have a limited applicability. At present, the most commonly used methods are the approximation-combinatorial method, whose applicability is determined by the type of the objective function and the size of the problem (see [1]), and cone methods, which critically depend on the dimension of the vector argument (see [2]). The latter are more universal, and their approbation suggests that the dimensionality problem can be solved using sufficiently powerful computers (see [2, 3]). However, in my attempts to use cone methods for optimizing on a feasible polyhedron of a high dimension (about 20), I discovered that these methods are numerically unstable. In this paper, a stable algorithm is proposed for numerical solution of the following maximization problem for a convex function defined on $E^m$:

$$f(x) \rightarrow \max,$$

$$\sum_{i=1}^{m} x_i = A,$$  

$$\bar{x}_i \leq x_i \leq \underline{x}_i, \quad i = 1, m,$$  

where $\bar{x}_i < \underline{x}_i$ for all $i$ and the feasible region determined by relations (2), (3) is nonempty and does not degenerate into a point.

We assume additionally that the function $f(x)$ is uniformly continuous on the set

$$U = \{ x : x \in P \text{ and } f(x) \leq f_{\max} + \Delta f_0 \},$$

where $P$ is hyperplane (2), $f_{\max}$ is the exact maximum of $f(x)$, and $\Delta f_0$ is the given tolerance for the computed value of the maximum.

The algorithm allows one to find an $\varepsilon$-optimal solution of this problem, i.e., a solution $x^{\text{opt}}$ such that $f(x^{\text{opt}}) \geq f_{\max} - \varepsilon$, where $\varepsilon = \Delta f_0$.

The assumption of the uniform continuity on $U$ is fulfilled for a broad class of problems. It is well known that a convex function on $E^m$ is continuous on $E^m$ (see [4]); therefore, (4) is a closed set. Suppose that this set is bounded (which is true for a broad class of functions $f(x)$); then, it is compact. A continuous function on a compact set is uniformly continuous on this set (see [5]). Other examples of classes of functions uniformly continuous on $U$ can also be given. Note, though, that the assumption of uniform continuity is used only for proving the finiteness of the optimization algorithm. Thus, uniform continuity is not obligatory in practical applications of this algorithm.
One has to deal with problem (1)–(3) in acceptance tests of engineering systems, namely, when a hypothesis on the variance of an objective characteristic is tested, based on a series of regression experiments (see [6, 7]).

A hyperparallelepiped (HP) of form (3) is denoted by $M^0$. An HP determined by bounds, such as in (3), is said to be correctly oriented. We use the notation $M_0$, $x_i$, and $\tilde{x}_i$ to refer not only to the HP defined by the constraints in the original problem but also to any correctly oriented HP embedded in the original one. Accordingly, we consider polyhedra in the hyperplane of the form $M = P \cap M^0$. The feasible region of problem (1)–(3) is one of such polyhedra. It will be clear from the context whether we deal with constraints (3) or a different correctly oriented HP. This extension of the concept of feasible polyhedron is related to the fact that, at each step of the proposed optimization algorithm, the bounds determining $M_0$ are modified to specify the domain $M$ that is not yet explored (not cut off).

2. PRINCIPLES OF THE THEORY

To justify the method for solving problem (1)–(3), we introduce a number of concepts and statements. The theorems are not proved, since the proofs are fairly simple and the analogous facts for polyhedra in the $m$-dimensional space are well known (see [8]).

A vertex of a polyhedron in the hyperplane is a point $x^0$ such that, for a certain positive integer $I$,

$$
\begin{align*}
  x_i^0 &= x_i \quad \text{or} \quad \tilde{x}_i \quad \text{for} \quad i \neq I, \\
  x_I^0 &= A - \sum_{i \neq I} x_i^0.
\end{align*}
$$

(5)

We say that a vertex is degenerate if $x_i^0 = x_i$ or $\tilde{x}_i$; otherwise, the vertex is nondegenerate. Note that, for a degenerate vertex, the index $I$ is not determined uniquely.

For a polyhedron in the hyperplane, an edge adjacent to a vertex $x^0$ is an $m$-dimensional segment of non-zero length with only two components varying:

$$
\begin{align*}
  x_i &= x_i^0 + \Delta, \\
  x_I &= x_I^0 - \Delta,
\end{align*}
$$

(6)

where $\Delta \geq 0$ when $x_i^0 = x_i$ and $\Delta \leq 0$ when $x_i^0 = \tilde{x}_i$, $i \neq I$. Feasible values of $\Delta$ are determined by constraints (3). Each nondegenerate vertex has $m - 1$ adjacent edges. Each edge has two vertices as its extreme points. One is the vertex appearing in the definition of the edge, and the other is called the adjacent vertex.

The simplest cone in the hyperplane is a point set determined by a system of $m - 1$ linearly independent vectors $B^{*}_i$ and a point $x_A$, called the vertex of cone, as

$$
  x = \sum_{i=1}^{m-1} \lambda_i B^{*}_i + x_A, \quad \lambda_i \geq 0.
$$

(7)

The edges of the simplest cone are the rays of the form $\lambda_i B^{*}_i + x_A$, where $\lambda_i \geq 0$.

**Theorem 1.** A polyhedron in the hyperplane is contained in the simplest cone that can be reconstructed from the nondegenerate vertex of the polyhedron and its adjacent edges.

A simplex is a portion of the simplest cone defined by an additional condition

$$
\sum_{i=1}^{m-1} \lambda_i \leq C, \quad C > 0.
$$

(8)

The vertices of a simplex in the hyperplane are $m$ points $x_A$ (the vertex of cone) and $x_A + CB^{*}_i$, $i = 1, m - 1$.

**Theorem 2.** A convex function attains its maximum on a simplex at a vertex of this simplex.

A similar assertion can easily be proved for a polyhedron $M$. 

COMPUTATIONAL MATHEMATICS AND MATHEMATICAL PHYSICS Vol. 43 No. 7 2003
A cutting hyperplane is a hyperplane of the form
\[ b^\top (x - x_A) = 1 \] (9)
such that the half-space
\[ b^\top (x - x_A) \leq 1 \]
cuts off simplex (8) from cone (7).

**Theorem 3.** For the cone with the vertex \( x_A \) given by (7) and the simplex defined by (8) with \( C = 1 \), the cutting hyperplane (9) is determined by the vector \( b \) with the components
\[ b_i = \sum_{k=1}^{m-1} B_{ki}^+ \] (10)
where \( B^+ \) is the pseudoinverse of the matrix
\[ B = (B_{*,1}, \ldots, B_{*,m-1}) \].

### 3. AN ALGORITHM FOR SOLVING THE PROBLEM

To simplify the description and substantiation of the algorithm, we introduce the following assumption: for each polyhedron \( M \) created in the algorithm, the vertices that are tested for the maximum by comparing the corresponding value of the objective function with the current highest value are nondegenerate.

One can ensure that this assumption is fulfilled by following the recommendations in [9]; namely, if a degenerate vertex is discovered, then a small correction is introduced into parameters of the problem (for example, the value of \( A \) in condition (2) is modified).

Below, we assume that only the value of the maximum of \( f(x) \) is of interest, while no maximizer is required. If one does need a maximizer \( x_{opt} \), then the assignment \( x_{opt} = x \) must be done each time when the best (record) value Rec = \( f(x) \) is updated.

The proposed algorithm is based on the branch and bound scheme. At each step of this scheme, the region \( M \) tested for the maximum is divided into three subsets (branching), namely,

1. a subset that is tested for the maximum first, using the internal cutting algorithm;
2. a subset analyzed by a recursive repetition of branching;
3. a subset that is analyzed last by returning to the beginning of the current step. The branching in this subset is unlikely.

Subset (2) is empty (nonempty) only simultaneously with subset (3).

At the current step, one carries out the analysis and cuts in subset (1). Then, one passes to the next step, which is the analysis of subset (2), if this subset is nonempty; otherwise, subset (3) of the preceding step is analyzed.

A graphical representation of an optimization step is given in Fig. 1. Vertex 0 depicts the original subset, whereas vertices 1, 2, and 3 correspond to its portions (in the order indicated above). Vertex 1 is shaded dashes, since it always is an end vertex. An example of optimization process depicted as a rooted tree is shown in Fig. 2. In this case, the root denotes preparatory operations before optimization begins. The end vertices of the tree are all vertices of the first type and also vertices of the second and third types if they correspond to empty subsets (such vertices are crossed out in Fig. 2). The initial vertices of all steps are numbered according to the order of steps in the algorithm.

Suppose that a subset of the second type is nonempty. Then, in passing to the next branching step (i.e., before the recursion), one sets bounds for the HP \( M^0 \), which is smaller than at the previous step and determines the region \( M \) to be tested for the maximum. At a certain step, \( M^0 \) becomes so small that no subsets of the second and third types are selected at the next step and no further branching occurs.

In the cutting algorithm, the value of the criterion is found at a vertex of the polyhedron \( M \) tested at the current stage (at a local maximizer of \( f(x) \) on \( M \)). This value is assigned to the variable Rec, the best value of the algorithm, if the preceding best value is surpassed. Then, for the vertex under analysis, a neighborhood in \( M \) is constructed where the criterion does not exceed the value Rec + \( \Delta f_0 \). This neighborhood is cut...
off from the set to be analyzed further, and the region not yet explored is represented in the standard form
\( M = P \cap M^0 \) with a modified ("contracted") HP.

The same algorithm is used for cutting off an "unpromising" branch, corresponding to the third subset in the current partition. Thus, while the cuts in the classical branch and bound scheme use an upper bound for the criterion on a subset unpromising for branching, the proposed method finds a region where the current record is safe from surpassing up to the given tolerance. If the unpromising set is contained in this region, then the corresponding branch is cut off; otherwise, a repeated branching is performed. A common feature in both methods is that the \( \varepsilon \)-optimality criterion is used to terminate the search (see [10]). This ensures the finiteness of the algorithm in difficult problems. In practice, this allows one to terminate the search in a reasonable amount of time.

The initial data for the internal cutting algorithm are as follows: the region \( M = P \cap M^0 \), determined by the "moving" bounds for \( M^0 \); the current highest value \( \text{Rec} = \text{Rec}_{\text{init}} \) (at the first step, we set \( \text{Rec} = -\infty \)); the parameter \( \Delta f_0 \); and the constant \( L_B \), which is chosen to be an order of magnitude greater than the diameter of the original HP (3). For example,

\[
L_B = 20 \max_i (\bar{x}_i - \underline{x}_i).
\]

The algorithm results in a new value \( \text{Rec} \geq \text{Rec}_{\text{init}} \) and a decision whether an uncut region is available. This region (if available) is represented as

\[
M' = P \cap M'^{\#}, \quad (M'^{\#}) = \{ x : \underline{x}_i \leq x_i \leq \bar{x}_i, \ i = 1, m \}. \tag{12}
\]

Then, the bounds for \( M^0 \) are modified so that \( M^0 = M'^{\#} \).

**Algorithm 1**

**Step 1.** Using the simplex method, a local maximizer \( x_A \) is found in form (5). If \( f(x_A) > \text{Rec} \), then we set \( \text{Rec} = f(x_A) \).

**Step 2.** The direction vectors for the edges of \( M \) adjacent to \( x_A \) are written out. According to (6), there are \( m - 1 \) vectors \( B_{ki}, i = 1, m, i \neq I \), with the components

\[
B_{ki} = 0, \text{ if } k \neq i \text{ and } k \neq I, \quad B_{ii} = -B_{ki} = \Delta_i.
\]

These vectors are renumbered so that the indices \( i \) increase from 1 to \( m - 1 \).

**Step 3.** Hyperplane (9) that cuts the simplex \( S \) from cone (7) in which the objective function does not exceed the value \( \text{Rec} + \Delta f_0 \) is constructed:

3.1. The support vectors of the cutting hyperplane are determined; i.e., for each vector \( B_{ki} \), the maximum scale factor \( C \) that satisfies the relations

\[
f(x_A + CB_{ki}) \leq \text{Rec} + \Delta f_0 \quad \text{and} \quad \| CB_{ki} \|_E \leq L_B \tag{13}
\]

is found. Then, the renormalization \( B_{ki} = CB_{ki} \) is carried out.
3.2. The pseudoinverse $B^+$ of matrix (11) is calculated. To this end, one deletes the lower row in $B$, inverts the resulting matrix by the Gauss method, and augments the inverse matrix with a zero column.

3.3. The components of the vector $b$ are calculated, using formulas (10).

**Step 4.** The presence of the uncut region $M \setminus S$ is verified:

4.1. Using the simplex procedure, the maximum

$$L_{\text{max}} = \max_{x \in M} b^T (x - x_A)$$

is found.

4.2. If $L_{\text{max}} \leq 1$, then $M \setminus S = \emptyset$, and the analysis of the region $M$ for the maximum is terminated: $\text{Rec} \geq \max f - \Delta f_0$.

**Step 5.** The bounds $x_i^+$ and $\bar{x}_i^+$ are determined for HP (12) that contains $M \setminus S$ and has the edges of the least length.

**Step 6.** The “contraction” parameter

$$d = 100 \max_i \frac{\bar{x}_i - \bar{x}_i^+ + \bar{x}_i^+ - \bar{x}_i}{\bar{x}_i - \bar{x}_i^+}$$

is evaluated. It determines to what extent (on a percentage basis) the size of the HP is reduced as a result of contraction.

**Step 7.** The assignments $\bar{x}_i = x_i$, $i = 1, m$, are carried out.

Explanation of the algorithm.

Step 3.1. The relation $f(x) \leq \text{Rec} + \Delta f_0$ for the simplex $S$ follows from the first inequality in (13) and Theorem 2. The second inequality in (13) is a safety measure for convex functions monotonically decreasing on a ray (see [4, p. 104]).

Step 3.2. The numerical stability in calculating $B^+$ is ensured by a specific form of the matrix $B$. By a proper normalization of the support vectors, $B$ can be reduced to the form

$$B^0 = \frac{1}{\sqrt{2}} \begin{bmatrix} -E & O \\ 1 & \ldots & 1 & -I \\ O & -E \end{bmatrix},$$

where $E$ are identity matrices of appropriate orders and 0 are zero rectangular matrices. It is easy to see that the angle $\alpha$ between an arbitrary column of $B$ and the span of other columns satisfies the inequality $|\sin \alpha| \geq \sqrt{2}/2$. Thus, no multiple contingency of the support vectors occurs (see [11]), which makes it possible to obtain the pseudoinverse numerically.

Step 3.3. The vector $b$ determines hyperplane (9), which ensures the elimination of solutions that are known not to be $\varepsilon$-optimal.

Step 4.2. The conclusion that $M \setminus S = \emptyset$ is made on the basis of Theorems 1 and 3.

**Step 5.** The region $M \setminus S$ can be represented by a system of linear constraints and the desired bounds as minima or maxima of corresponding coordinates; i.e., each bound is a solution to a linear programming problem. The HP of form (12) determined at this step is thereafter called the contracted HP.

**Step 6.** In what follows, the parameter $d$ is used for checking whether a repeated cut makes sense after the region $M$ has been modified. A repetition of the algorithm is inexpedient when $d \sim 1$.

Since Algorithm 1 is an internal procedure in the branch and bound scheme, the fact that the analysis of the current region at Step 4.2 is terminated may signify cutting off the corresponding branch and passing to a new region $M$.

After the first of the three subsets selected at a branching step has been analyzed, we have to solve the maximization problem for $f(x)$ on the region $M$ with an $M^0$ of reduced size. The second and the third subsets are constructed by dividing the HP obtained into two halves:

$$M^0_i = \{ x : x_k \leq x_i \leq \bar{x}_k \ \forall k \neq i, \ x_i \leq x_l \leq (x_i + \bar{x}_i)/2, \ \text{if} \ l = 1, \ \text{or} \ (x_i + x_l)/2 \leq x_l \leq \bar{x}_l, \ \text{if} \ l = 2 \},$$

$$i = 1, m, \ l = 1, 2.$$
Accordingly, the second and the third subsets have the forms \( P \cap M_{i_1}^0 \) and \( P \cap M_{i_2}^0 \). This feature explains the fact that Algorithm 2, presented below, is called the bisection algorithm. The algorithm uses the variable \( N \), the number of a recursive call. If the analysis of the region \( M \) is terminated at Step 4.2 of Algorithm 1, then the following operations have to be executed:

- if \( N > 0 \), then go to Step 9 of Algorithm 2 (see below);
- if \( N = 0 \), then terminate. The maximization problem (1)–(3) is solved, and the maximum value of the function is equal to Rec.

In formulating the bisection algorithm, we use the concept of a stack, which is a memory region where one stores values of certain variables prior to a recursive call. After the recursion has been completed, the stack is used to extract the values of these variables that were last put there.

**Algorithm 2**

**Step 1.** Set \( N = 0 \).
**Step 2.** Set \( \text{Rec} = -\infty \).
**Step 3.** Execute Algorithm 1.
**Step 4.** If \( d > 3 \), then go to Step 3.
**Step 5.** Set \( N = N + 1 \).
**Step 6.** Set the index \( i_0 \) specifying the partition of \( M^0 \) into two halves \( M_{i_1}^0, l = 1, 2 \). Set the indices \( l_1 \) and \( l_2 \) for the halves that are analyzed for the maximum in the first and the second place, respectively.

**Step 7.** Assign the bounds of \( M_{i_l}^0 \) to the variables \( \bar{x}_i \), \( i = 1, m \) (in particular, \( \bar{x}_i = \bar{x}_i \) if \( i \neq i_0 \)).

**Step 8.** Maximization of \( f(x) \) inside \( M_{i_l}^0 \):

1. Modify one of the bounds \( \bar{x}_{i_0} \) or \( \bar{x}_{i_0} \) according to (15) with \( l = l_1 \) and \( i = i_0 \).
2. Execute the contraction procedure; i.e., execute the stand-alone Step 5 of Algorithm 1. (The required vector \( b \) and vertex \( x_A \) were earlier calculated at Step 3 of the current algorithm.)
3. Set \( \bar{x}_i = \bar{x}_i \) (\( i = 1, m \)) and \( \text{Rec1} = \text{Rec} \).
4. Put the variables \( \bar{x}_i \) (\( i = 1, m \)), \( b_i \) (\( i = 1, m \)), and \( \text{Rec1} \) in the stack.
5. Go to Step 2 (recursion).

**Remark.** The transition to the next step is possible only from Step 4.2 of Algorithm 1. It signifies the termination of the current recursion.

**Step 9.** Data preparation for the maximization inside the HP \( M_{i_l}^0 \):

1. Extract the variables \( \bar{x}_i \) (\( i = 1, m \)), \( b_i \) (\( i = 1, m \)), and \( \text{Rec1} \) from the stack.
2. If \( \text{Rec1} > \text{Rec} \), then set \( \text{Rec} = \text{Rec1} \).
3. Set \( \bar{x}_i = \bar{x}_i \) (\( i = 1, m \)).

**Step 10.** Maximization of \( f(x) \) inside \( M_{i_l}^0 \):

1. Set \( N = N - 1 \).
2. Go to Step 3.

At Step 8, we essentially repeat the solution of the original problem with a reduced diameter of the feasible region. By a recursive repetition of this step, we reduce the diameter of the feasible region until it is possible to find the maximum in this region by repeated execution of Algorithm 1 (according to Step 3 above). This terminates the current recursion.

4. FINITENESS AND ACCELERATION OF THE ALGORITHM

We show that, under certain rules for choosing index \( i_0 \), it is possible to bound the length of branches beginning in the root of the optimization graph at Step 6 of Algorithm 2. This ensures that Algorithm 2 can
be executed in a finite number of operations. In what follows, the Euclidean norm of a matrix (a vector) is denoted by \( \| \|_E \).

**Theorem 4.** Let a convex function \( f(x) \) be uniformly continuous on region (4). Then, there exists a \( \delta > 0 \) such that, for any polyhedron \( M = P \cap M^0 \) and any vertex \( x_A \) found at Step 1 of Algorithm 1, all points \( x \in M \) satisfying the relation
\[
\| x - x_A \|_E \leq \delta
\]
also belong to the simplex \( S \) constructed at Step 3 of Algorithm 1.

**Proof.** Uniform continuity means that it is possible to find an \( \epsilon' > 0 \) such that, for the points in \( U \), one has
\[
| f(x') - f(x'') | \leq \Delta f / 2
\]
everywhere \( \| x' - x'' \|_E \leq \epsilon' \). However, for even scale factor \( C \) determined at Step 3.1 of Algorithm 1, both \( x_A \) and \( x_A + CB_{*} \) belong to \( U \). If the first inequality in (13) becomes an equality, then, upon renormalization, we obtain \( \lambda \geq \epsilon' \). In the general case, the normalized vectors \( B_{*} \) certainly satisfy the relation
\[
\| B_{*} \|_E \geq \epsilon = \min(\epsilon', L B)
\]
and
\[
\sum_{i=1}^{m-1} \lambda_i \leq \sqrt{m} \| \lambda \|_E \leq \sqrt{m} \| N_B^{-1} (B^0)^+ \|_E \| x - x_A \|_E \leq (m-1) \epsilon^{-1} \| (B^0)^+ \|_E \delta \leq (m-1) \epsilon^{-1} Q_m \delta,
\]
where \( Q_m = \max \{ \| (B^0)^+ \|_E \} \) and the maximum is taken over \( I = \{1, m\} \).

Thus, one can find a \( \delta \) which guarantees that inequality (8) with \( C = 1 \) is fulfilled. By Theorem 3, this proves that the vector \( x \) belongs to the simplex \( S \). The theorem is proved.

It is evident that, if the procedure for choosing \( i_0 \) in Algorithm 2 ensures that the diameter of \( M^0 \) can be made arbitrary small as \( N \rightarrow +\infty \), then there exists an \( N_0 \) such that, for all \( N \geq N_0 \) and \( x \in M \), inequality (16) is fulfilled. However, in view of Theorem 4, this means that the corresponding branch will be cut off at Step 4 of Algorithm 1. Therefore, the bisection algorithm terminates in a finite number of steps.

The efficiency of the branch and bound scheme is to a considerable extent determined by the branching method. In our case, this is the rule for choosing the indices \( i_0, l_1, \) and \( l_2 \) at Step 6 of Algorithm 2. We can recommend the following scheme for choosing these parameters.

For given \( M^0 \) and cutting hyperplane (9), we associate with each polyhedron \( M_{il} = P \cap M^0_{il} \) the contracted HPs (12). Denote their volume by
\[
V_{il} = \prod_{i=1}^{m} (\bar{x}_i - \bar{x}_i^1).
\]

We propose to set
\[
i_0 = \arg \max_i (\bar{x}_i - \bar{x}_i), \quad l_2 = \arg \min_{i=1,2} V_{il}.
\]

This criterion ensures that the halves of the HP have a minimum diameter after the partition, because the index \( i_0 \) corresponds to the coordinate along which \( M^0 \) has the largest extension. Moreover, the part with a smaller volume is inspected in the second place. In this way, we bound the maximum length of branches in the optimization graph, on the one hand, and ensure that unpromising branches are efficiently cut off, on the other hand.
5. APPROBATION RESULTS

We can draw certain conclusions on the efficiency of the bisection algorithm on the basis of solving various problems of form (1)–(3). Below, these conclusions are formulated and explained, using the following test example:

\[
f(x) = \sum_{i=1}^{m} \left[ (m - i + 1)x_i + \frac{\sqrt{i x_i}}{m} \right]; \quad x_i = 0, \quad \bar{x}_i = w = 1, \quad i = 1, \ldots, m, \quad A = m/2, \quad \Delta f_0 = 0.5 \quad (17)
\]

(in some experiments, the parameters \( A \) and \( w \) were varied).

All results presented below were obtained on an AT-compatible computer with a 200 MHz WinChip C6 processor.

**Conclusion 1.** For a given dimension \( m \), the total computational work in the bisection algorithm can significantly vary as the parameters \( x_i, \bar{x}_i, A \), and \( \Delta f_0 \) in the problem vary.

To substantiate this conclusion, we depict approximate dependences of the time for solving the test problem on the parameter \( A \) for \( m = 20 \) and \( m = 40 \) in Fig. 3. All points (except one shown in the plot for \( m = 20 \)) belong to the fitting curves. For comparison, we also present here the total number of branching vertices \( n_{gr} \) in the optimization graph as a function of \( A \). Figure 4 shows how the optimization time depends on the length \( w = \bar{x}_i - x_i \) of the interval where the variables vary for \( m = 40 \) and \( A = 20 \). When \( w > 1.2 \), the fitting function decreases, which is in agreement with the curves in Fig. 3. A special feature of problem (17) is that the optimization time \( T(A) \) attains its maximum at \( A = 0.5\Sigma(x_i + \bar{x}_i) \) for \( i = 1, \ldots, m \) and decreases rapidly when \( A \) deviates from this number. This results in that the computation time decreases when \( w \) grows substantially, while \( A \) is fixed.

**Conclusion 2.** For a certain class of problems of type (1)–(3), the optimization time in the bisection algorithm is a slower growing function of \( m \) than the time in the exhaustive search method.

Here, we mean that, for a certain class of problems, the computation time \( T \) in the new optimization method behaves approximately as \( T \sim (m - c)^\alpha \), where \( \alpha > 0 \), while a typical time dependence for the exhaus-

<table>
<thead>
<tr>
<th>( m )</th>
<th>( n_M )</th>
<th>( m )</th>
<th>( n_M )</th>
<th>( m )</th>
<th>( n_M )</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>22176</td>
<td>16</td>
<td>411840</td>
<td>20</td>
<td>7390240</td>
</tr>
<tr>
<td>13</td>
<td>24024</td>
<td>17</td>
<td>437580</td>
<td>21</td>
<td>7759752</td>
</tr>
<tr>
<td>14</td>
<td>96096</td>
<td>18</td>
<td>1750320</td>
<td>22</td>
<td>31039008</td>
</tr>
<tr>
<td>15</td>
<td>102960</td>
<td>19</td>
<td>1847560</td>
<td>23</td>
<td>32449872</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( m )</th>
<th>( T ), min</th>
<th>( f_{max} )</th>
<th>( n_{gr} )</th>
<th>( N_{max} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>2.88</td>
<td>127.1</td>
<td>63</td>
<td>8</td>
</tr>
<tr>
<td>21</td>
<td>8.32</td>
<td>171.6</td>
<td>109</td>
<td>10</td>
</tr>
<tr>
<td>23</td>
<td>14.4</td>
<td>205.2</td>
<td>145</td>
<td>11</td>
</tr>
<tr>
<td>26</td>
<td>27.4</td>
<td>261.3</td>
<td>182</td>
<td>12</td>
</tr>
<tr>
<td>29</td>
<td>51.0</td>
<td>323.8</td>
<td>229</td>
<td>13</td>
</tr>
<tr>
<td>32</td>
<td>72.8</td>
<td>393.4</td>
<td>231</td>
<td>15</td>
</tr>
<tr>
<td>35</td>
<td>102.4</td>
<td>469.4</td>
<td>226</td>
<td>16</td>
</tr>
<tr>
<td>38</td>
<td>132.0</td>
<td>552.5</td>
<td>237</td>
<td>17</td>
</tr>
<tr>
<td>41</td>
<td>151.5</td>
<td>642.0</td>
<td>191</td>
<td>18</td>
</tr>
<tr>
<td>44</td>
<td>199.0</td>
<td>738.6</td>
<td>215</td>
<td>20</td>
</tr>
<tr>
<td>47</td>
<td>194.1</td>
<td>841.6</td>
<td>165</td>
<td>19</td>
</tr>
<tr>
<td>50</td>
<td>232.2</td>
<td>951.7</td>
<td>152</td>
<td>22</td>
</tr>
</tbody>
</table>
tive search method is $T \sim a^m$, $a > 1$. We have chosen the exhaustive search method for comparison, because, until recently, only this method remained usable for $m$ close to 20.

The results of solving the test problem by the two methods being compared are shown in Fig. 5. The fitting function is

$$T = (1.807 \times 10^{-5}) \times 2.154^m \text{ min}$$

for the exhaustive search method and

$$T = (5.37 \times 10^{-3})(m - 10)^{3.04} \text{ min}$$

for the bisection algorithm.

The exhaustive search is preferable when $m < 14$. However, the fast growth of the number of vertices $n_M$ in the feasible polyhedron as a function of $m$ results in that the exhaustive search optimization slows down catastrophically when $m > 21$ (see also Table 1). It should be noted though that, for a given $m$, the number $n_M$ strongly depends on the parameters of the feasible region. Thus, Conclusion 2 is not universal.

The calculations conducted for the construction of curve 2 in Fig. 5 are selectively shown in Table 2. Similar to Fig. 3, one can observe the dependence of the computation time $T$ on the total number of branching vertices $n_{gr}$ in the optimization graph and, in addition, its dependence on the parameter $N_{\text{max}}$, the maximum recursion depth in the algorithm.

When constructing curves 1 and 2 in Fig. 5, no distinctions were discovered between the maximization results obtained by the bisection algorithm (curve 1) and the exhaustive search method (curve 2). Moreover, the point $x_4$ found in the first application of the simplex method (Step 1 in Algorithm 1) turned out to be a global maximizer. A similar situation was observed in the applied problem of inspecting specimens in the space industry (see [6, 12]). This made it possible to interpret a local maximizer as an approximation to the global one with the resulting acceleration in decision making (see [12]). However, in tuning the new algorithm, we came across situations when local and global solutions were different in practical problems (see [12]) as well as in test problems of type (17). In example (17) with the parameters $m = 6$, $w = 20$, and $A =
The first local optimizer was $x_A = (0.1, 20, 0, 0, 20, 0)^T$ with $f(x_A) = 384.0$, whereas the global optimizer was $x^{opt} = (0.1, 20, 20, 0, 0, 0)^T$ with $f_{max} = 390.4$. The optimization time was 80 s.

The dependence between the storage used and the size of the test problem (17) shows that even problems with large $m$ can be solved using only 640 Kb of RAM.

REFERENCES