A general phase-I method in linear programming

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computing Science

Memorandum COSOR 83 - 01

A general Phase-I method in linear programming

by

Istvan Maros

Eindhoven, the Netherlands

January 1983

A GENERAL PHASE-I METHOD IN LINEAR PROGRAMMING*

Istvan Maros**

1. Introduction

The basic technique for solving LP problems is still the simplex method [2]. It has many variants but in practice the primal simplex methods are considered the most important. Phase I of the primal methods serves for finding a basic feasible solution but the same procedure can also be used for generating feasible points of other problems with linear constraints, or even for checking the consistency of a system of linear equalities/inequalities. Since in Phase I the objective function is not or only slightly considered it usually does not move towards optimality. It would be advantageous if Phase I were as short as possible or if it could better take into account the real objective function. In this paper we try to contribute to both of these aspects positively.

---

* This paper is based on a contributed paper of the author at the International Workshop on Advances in Linear Optimization Algorithms and Software, Pisa, July 1980. At the meeting the author's attention was brought to an early publication [10] of Philip Wolfe where he first proposed the idea of dropping condition 1.2.

** Computer Application Company, P.O. Box 146, H-1502 Budapest, Hungary; presently at Dept. of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600MB Eindhoven, the Netherlands.

The traditional Phase I methods work under the following conditions:

1.1 The incoming variable takes a feasible value.

1.2 The feasible basic variables remain feasible after a transformation (the number of infeasibilities does not increase).

1.3 The outgoing variable leaves the basis at a feasible level (lower or upper bound).

These three conditions still do not determine the outgoing variable uniquely for a given incoming variable. If we relax condition 1.2 further possibilities arise. One of them is presented in this paper. At each iteration this algorithm maximizes the progress towards feasibility with the selected incoming variable simply by a new rule for determining the outgoing variable at the expense of very little extra computational work.

The idea of determining the outgoing variable is combined with an adaptive column selection strategy for determining the incoming variable. The technique is able dynamically to take into account the objective function. This special composite Phase I procedure requires extra computational work but this appears to be acceptable because of the significant reduction in the number of iterations and in the overall computational effort necessary to obtain the answer to an LP problem.

The described procedure has been implemented in the LP package LIPROS (this is an LP package of the R-10 small computers produced in Hungary). The computational experiences are favourable but the algorithm still requires further testing and validation.

The improvement in the performance is due to

- a sharper reduction of the sum of the infeasibilities
- efficient steps in the presence of degeneracy
- improved numerical stability.

As a result of the adaptive composite Phase I strategy usually very few iterations are required for Phase II, in other words the search for feasibility and optimization are done simultaneously. In Section 10 a generalization of the procedure is presented.

2. Problem statement

The present day LP packages generally use the upper bounding technique. For easy reference in the sequel we shall consider the following LP problem.

$$(2.1) \qquad \bar{y}_i + \sum_{j=1}^{n} \bar{a}_{ij}\bar{x}_j = b_i \qquad i = 1,\dots,m$$

or in matrix form

$$(2.2) \qquad Ax = b \ , \text{ where } A = (I\,,\bar{A}) \qquad x = \begin{pmatrix} \bar{y} \\ \bar{x} \end{pmatrix}$$

and any one of the rows of A can be defined as the objective function. The $\bar{x}_j$-s are called structural variables and the $\bar{y}_i$-s are called logical variables. The logical variable of the objective row is maximized. In addition to constraint (2.2) variables have individual bounds of their feasibility ranges on the basis of which they are divided into 4 types:

(2.3)

type	feasibility range
0	$x_j = 0$
1	$0 \le x_j \le u_j \neq 0$
2	$0 \le x_j \le +\infty$
3	$-\infty \le x_j \le +\infty$

.

It can be seen that by simple transformations any LP problem can be brought into this form. The transformations can be carried out during the input of the problem.

Let us denote by B a basis to the (2.2) system and by β the corresponding solution:

(2.4) $B\beta = \bar{b}$ or $\beta = B^{-1}\bar{b}$.

Here $\bar{b}$ is the right-hand-side adjusted to account for the non-basic variables at upper bound:

$$\bar{b} = b - \sum_{j \in J} u_j a_j$$

where J is the index of such variables and a_j is the j-th column of (2.2).

We need further notation:

I_B is the index set of basic variables,		
I_0	- " -	of type 0,
I_1	- " -	of type 1,
I_2	- " -	of type 2,
and I_3	- " -	of type 3.

Consequently:

(2.5) $$I_B = I_0 \cup I_1 \cup I_2 \cup I_3 .$$

If x_j is of type 0 then its upper bound is defined as $u_j = 0$. The index set of upper bounded variables will be denoted by I_u:

$$I_u = I_0 \cup I_1 .$$

We divide the basic variables into three classes according to their feasibility status:

$$M = \{i : i \notin I_3 \wedge B_i < 0\}$$

$$P = \{i : i \in I_u \wedge B_i > u_i\}$$

$$F = I_B \setminus (M \cup P) .$$

M is the index set of those variables which are infeasible in the minus direction, P is the index set of the variables infeasible in the plus direction and F is the index set of the feasible variables. A type 3 basic variable is always feasible and therefore belongs to F.

The measure of infeasibility of a basic solution is defined as

(2.6) $$w = \sum_{i \in M} \beta_i - \sum_{i \in P} (\beta_i - u_i) .$$

This definition is similar to that of Orchard-Hays [9] and can be interpreted as the negative of the sum of the violations. It is evident that $w \leq 0$. If $w = 0$ then both M and P are void and the solution is feasible. Therefore Phase I of the simplex method is to

(W)
maximize w
subject to (2.2) and (2.3) .

To solve problem W the basic technique of the simplex method can be used, though it is not a conventional LP problem since the composition of the objective function changes as the sets M and P change.

3. Discussion of problem W

This discussion assumes that condition 1.2 is satisfied. Let us suppose that a nonbasic variable x_j is at level 0. We try to increase its value to $t > 0$ and want to know its effect on w. To maintain the basic equalities of (2.4) the values of the basic variables change as a function of t.

This functional relation is denoted by f(t) giving:

(3.1) $$Bf(t) + ta_j = \bar{b}$$

or

(3.2) $$f(t) = \beta - t\alpha_j$$

where α_j is the updated column: $\alpha_j = B^{-1}a_j$.

The coordinate form of (3.2) is

(3.3) $$f_i(t) = \beta_i - t\alpha_{ij} .$$

If we are at a basis B for which $w < 0$ the following lemma holds.

Lemma 3.1. w can be improved by increasing x_j only if

(3.4) $$d_j = \sum_{i\in M} \alpha_{ij} - \sum_{i\in P} \alpha_{ij} < 0 .$$

Proof. The proof is straightforward. Let us suppose that $t > 0$ is small enough so that sets M and P remain unchanged. In this case the change of w is given by

$$\Delta w = \sum_{i\in M} [f_i(t) - f_i(0)] - \sum_{i\in P} \{[f_i(t) - u_i] - [f_i(0) - u_i]\} =$$

$$= \sum_{i\in M} (-t\alpha_{ij}) - \sum_{i\in P} (-t\alpha_{ij}) =$$

$$= -t\left(\sum_{i\in M} \alpha_{ij} - \sum_{i\in P} \alpha_{ij} \right) = -td_j \, .$$

Hence $d_j < 0$ is necessary for $\Delta w > 0$.

If sets M and P remain unchanged only for $t = 0$ then the basis is degenerate and - because of condition 1.2 - x_j could only enter basis at the level zero.

If a nonbasic variable moves in a negative direction (a type 1 variable coming in from upper bound or a type 3 variable entering with negative value) then $d_j > 0$ is necessary for $\Delta w > 0$. □

Step one of an iteration in Phase I is checking the d_j values of the non-basic variables. There may be many candidates with the correct sign of d_j and a decision must be made to select one of them. This selection heavily influences the number of iterations needed to obtain a solution to the LP problem (2.1). Some details of the selection (called pricing) are discussed in section 8.

Now we suppose that the incoming variable has been selected and we want to investigate the movements of the basic variables as a function of the magnitude of the incoming variable. For simplicity we drop index j from (3.3) and use the form

$$f_i(t) = \beta_i - t\alpha_i \,. \tag{3.5}$$

We are interested in the changes of the feasibility status of each of the basic variables and therefore exclude type 3 variables from further considerations and restrict ourselves to the set

$$I = I_B \setminus I_3 \,.$$

A variable in I is negative-infeasible if its value is negative and a variable in I_u is positive-infeasible if its value is greater than its upper bound. Using the following notation

$$Z^- = \begin{cases} 0 & \text{if } Z \geq 0 \\ Z & \text{if } Z < 0 \end{cases}$$

$$Z^+ = \begin{cases} Z & \text{if } Z > 0 \\ 0 & \text{if } Z \leq 0 \,, \end{cases}$$

the measure of infeasibility as a function of t can be expressed in the following way:

$$\begin{aligned} w(t) &= \sum_{i \in I} [f_i(t)]^- - \sum_{i \in I_u} [f_i(t) - u_i]^+ = \\ &= \sum_{i \in I} [\beta_i - t\alpha_i]^- - \sum_{i \in I_u} [\beta_i - t\alpha_i - u_i]^+ \,. \end{aligned} \tag{3.6}$$

From this form it is clear that w(t) is a continuous and piecewise linear function. It has break points at the t values where the feasibility status of a least one of the variables changes. It is also evident, that for $t = 0$ w(t) gives the w of (2.6). In this sense w(t) can be considered as a function-wise extension of w.

In (3.6) $f_i(t)$ contributes to the first sum while $f_i(t) < 0$, that is while $i \in M$. Similarly for $i \in I_u$, $f_i(t) - u_i$ contributes to the second sum of (3.6) while $i \in P$. The feasibility status of a variable changes when it crosses a boundary of its feasibility range. We denote by T_{ℓ_i} the value of t at which the variable i reaches its lower bound (equal to zero) and by T_{u_i} the value of t at which the variable i reaches its upper bound in the case of $i \in I_u$.

Since we are interested in non-negative values of t (the incoming variable moves in a positive direction) the break points are defined by

$$\left.\begin{array}{lll} (3.7) & T_{\ell_i} = \beta_i/\alpha_i > 0 & (\alpha_i \neq 0) \\ \text{or} & T_{\ell_i} = 0 \ \text{ if } \ \beta_i = 0 \wedge \alpha_i > 0 & \end{array}\right\} i \in I$$

and

$$\left.\begin{array}{lll} (3.8) & T_{u_i} = (\beta_i - u_i)/\alpha_i > 0 & (\alpha_i \neq 0) \\ \text{or} & T_{u_i} = 0 \ \text{ if } \ \beta_i = 0 \wedge \alpha_i < 0 & \end{array}\right\} i \in I_u$$

If $u_i = 0$ then $T_{\ell_i} = T_{u_i}$. Of course there are many cases when some of the T_{ℓ_i} and T_{u_i} values may be identical, that is the break points may have "multiplicity". Figures 1 and 2 give examples of how the break points are defined. Fig. 1 Fig. 2

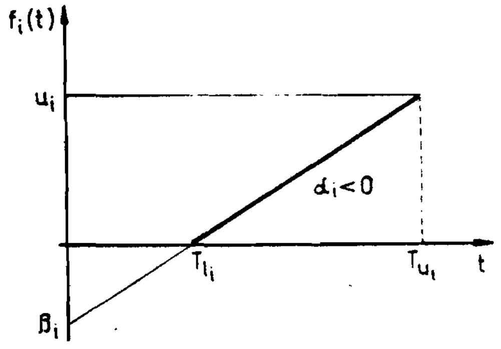


fig. 1.

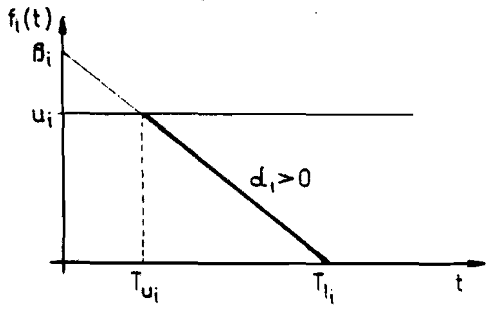


fig. 2.

4. Discussion of w(t)

Now we relax condition 1.2 and allow feasible basic variables to become infeasible. We want to determine a value for the incoming variable which maximizes w(t). It will be shown that the function w(t) has some desirable properties so that such a maximization can easily be carried out. This maximization is achieved by a simplex-type basis change.

Recall that the value of the i-th basic variable as a function of t is expressed by

$$f_i(t) = \beta_i - t\alpha_i$$

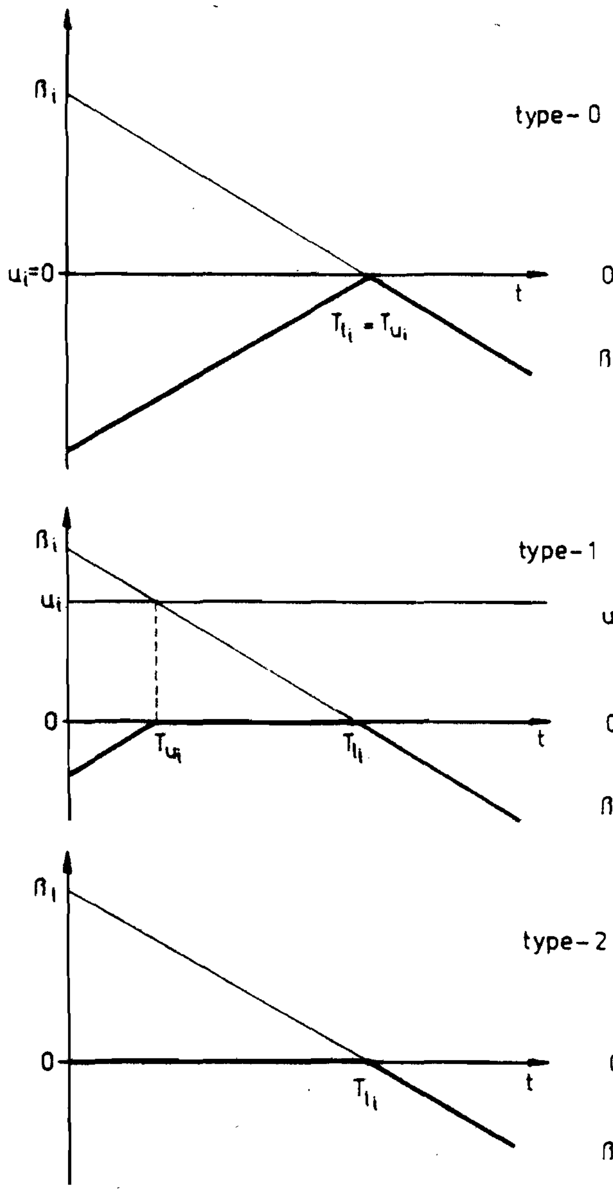
and therefore the feasibility of the i-th basis variable depends on its type, the value of β_i and the sign of α_i. The contribution of $f_i(t)$ to w(t) is illustrated in Figures 3 (where $\alpha_i > 0$) and 4 (where $\alpha_i < 0$). The function $f_i(t)$ is represented by a thin line and its contribution to w(t) by a thick line. The values T_{ℓ_i} and T_{u_i} are those defined in (3.7) and (3.8).

ig. 3
ig. 4

All other cases can be derived as special cases of one of these figures.

The contribution of $f_i(t)$ to w(t) is in all cases a concave function. Since w(t) is the sum of these contributions it is also - as a sum of concave functions - a concave function.

Now we are ready to discuss w(t), and consider how it behaves at the break points. The first change in the feasibility status of one of the basic variables occurs when the value of t reaches the smallest value among those defined by (3.7) and (3.8). For use later these values are sorted into as-



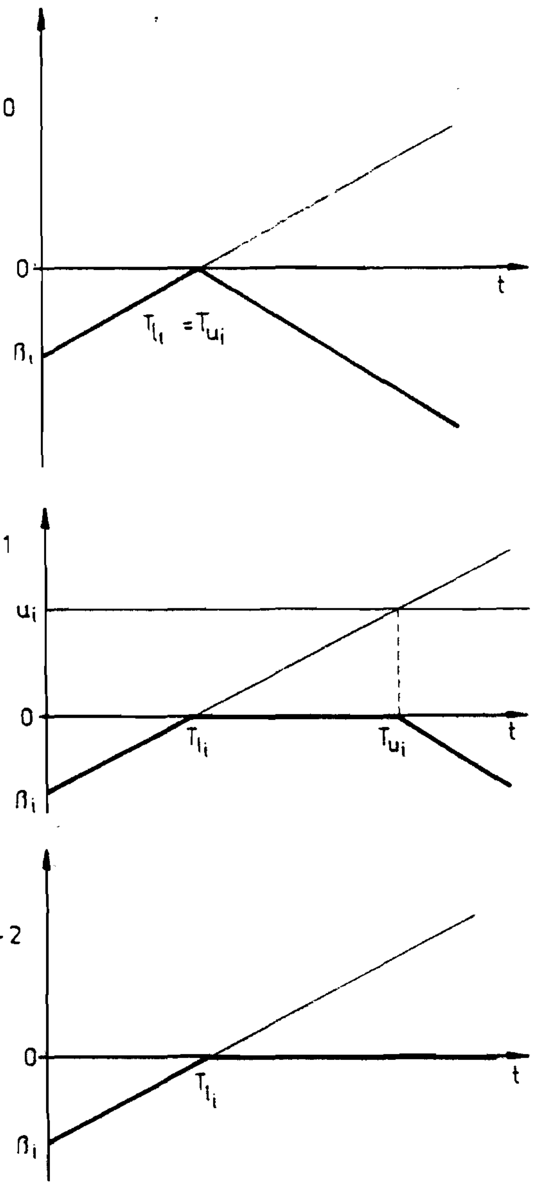


fig.3. $\alpha_i > 0$

fig.4. $\alpha_i > 0$

cending order:

(4.1) $$0 \le t_1 \le \dots \le t_Q$$

where Q denotes the number of the defined values.

Since we are in Phase I it is true that $w(0) < 0$. Increasing the value of t from 0 to t_1 means that $w(t)$ increases at a rate of

(4.2) $$r_1 = -\left(\sum_{i \in M} \alpha_i - \sum_{i \in P} \alpha_i\right)$$

giving an increase of $r_1 t_1 \ge 0$ because $r_1 = -d_j$, see Lemma 3.1. At $t = t_1$ the feasibility status of at least one of the variables changes. Let the variable which defines t_1 be denoted by i_1. If $\alpha_{i_1} < 0$ then $f_{i_1}(t)$ reaches one of its bounds from below thus index i_1 moves either from M to F or (if $i_1 \in I_u \cap F$) from F to P. In both cases, from t_1 onwards, the slope of $w(t)$ will be $r_2 = r_1 + \alpha_{i_1}$ as can be seen from (4.2). Since α_{i_1} is negative this means that the slope of $w(t)$ decreases by $|\alpha_{i_1}|$ that is $r_2 = r_1 - |\alpha_{i_1}|$. If $\alpha_{i_1} > 0$ then $f_{i_1}(t)$ reaches one of its bounds from above thus index i_1 moves either from P to F of from F to M. In both cases - as can be seen also from (4.2) - the slope of $w(t)$ will be $r_2 = r_1 - \alpha_{i_1}$ which can be rewritten as $r_2 = r_1 - |\alpha_{i_1}|$. Thus we have obtained that in any case the slope of $w(t)$ decreases by $|\alpha_{i_1}|$. Since the same arguments are valid for any t_k $(k = 2,\dots,Q)$ it is true that at any break point the slope of $w(t)$ decreases by $|\alpha_{i_k}|$ giving $r_{k+1} = r_k - |\alpha_{i_k}|$. Thus we have proved the following:

Theorem 4.1. w(t) is a piecewise linear concave function with break points t_k (k = 1,...,Q) as defined in (4.1) and the slope of its (k + 1) st interval is

$$r_{k+1} = r_k - |\alpha_{i_k}| \qquad k = 1,\dots,Q\,.$$

In the case of coinciding points the length of an interval may be zero. Note that in the course of the proof the uniqueness of the t_k values was not exploited.

Some characteristic shapes of w(t) can be seen in Figures 5, 6 and 7.

'ig. 5
'ig. 6
'ig. 7

Figure 5 shows a typical curve. In Figure 6 w(t) reaches its theoretical maximum. Figure 7 shows an interesting phenomenon. A value of $t_1 = 0$ means that we are at a degenerate basis (since either $t_1 = \beta_{i_1}/\alpha_{i_1} = 0$ or $t_1 = (\beta_{i_1} - u_{i_1})/\alpha_{i_1} = 0$) and the value $|\alpha_{i_1}|$ is such that $r_2 = r_1 - |\alpha_{i_1}| <$ < 0, giving that $w(t) < w(0)$ for any $t > 0$, that is degeneracy and the magnitude of $|\alpha_{i_1}|$ prevent an increase in w(t). This case will be generally analysed in Section 5.

When attempting to maximize w(t) we have to take condition 1.1 into account. It requires that the incoming variable has to be feasible. If the incoming x_j variable is of type-1 it may happen that for the t_k point which maximizes w(t), $t_k \geq u_j$ will hold. This can be considered as a favourable special case from the viewpoint of the iterations since now we don't go to the maximum of w(t) but stop at $t = u_j$ and make an iteration without changing the basis simply by putting x_j to its upper bound. At such an iteration no new eta vector [9] is generated which is computationally very favourable and therefore such steps - called type-1 iterations - are preferred. Keeping in mind this situation the following theorem can be stated:

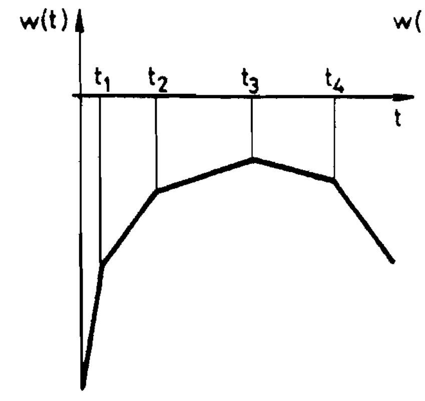


fig.5.

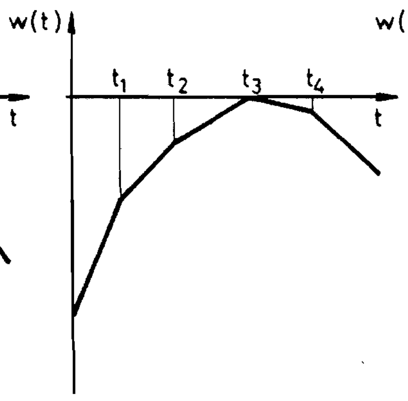


fig.6.

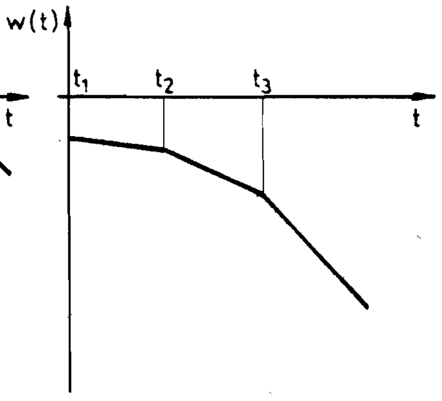


fig.7.

Theorem 4.2. If we are not faced with a type-1 iteration then $w(t)$ attains its global maximum at a break point $t = t_q$ and this defines such a change of the basis where the outgoing i_q-th variable leaves the basis at a feasible level and the incoming variable enters at a feasible value.

The first part of the theorem follows from the fact that $w(t)$ is a piecewise linear concave function (Theorem 4.1) while the second part is true because each t_k defines a change of the basis and the outgoing variable has a feasible value at t_k, since t_k is identical with either a T_{ℓ_i} or a T_{u_i} value and at these points the corresponding basic variable is always at a feasible level [see (3.7) and (3.8)].

Having determined points t_k and the corresponding values we easily can find the global maximum of $w(t)$, since it is reached at a point where the sign of the slope of $w(t)$ changes. This can be formulated by the following lemma:

Lemma 4.1. Set $r_1 = -d_j > 0$ and compute the following recursion:

(4.3) $$r_{k+1} = r_k - |\alpha_{i_k}| \qquad k = 1,2,\dots .$$

The maximum of $w(t)$ is defined by index q for which

(4.4) $$\begin{aligned} r_q &> 0 \\ r_{q+1} &\le 0 \end{aligned}$$

holds.

From now on q will be used to index that t_k value which defines the maximum of $w(t)$ in the sense of Lemma 4.1.

For use later we still need the optimum value of $w(t)$. This also can be computed by a simple recursion.

Lemma 4.2. Set $t_0 = 0$ and compute

$$(4.5) \qquad w(t_k) = w(t_{k-1}) + (t_k - t_{k-1})r_k \qquad k = 1,\dots,q .$$

$w(t_q)$ will be the required value as can easily be verified on the basis of Theorem 4.1 and the definition of t_k and r_k.

Note that if we take $q = 1$ we obtain the traditional method for determining the outgoing variable. In this respect the described method really can be condidered a generalization of the traditional one. This means that at any time when we have $q > 1$ we make a locally stronger iteration.

If the incoming variable enters the basis moving in a negative direction then we simply substitute $-\alpha_i$ instead of α_i in the previous discussion and everything remains valid.

Since from this procedure DEcent Leaps can be expected in PHase I, for further reference we shall call it DELPHI.

5. Degeneracy

A basis is degenerate if at least one of the basic variables is at one of its bounds. The danger of a degenerate basis is that the basic variables at bound with the "wrong" sign for α_i may block the incoming variable which can hence only enter the basis at level zero so making no progress towards feasibility/optimality and creating the possibility of cycling. Such a blocking can particularly easily occur with the traditional Phase I methods which satisfy condition 1.2.

The situation may be different if we relax this condition and use DELPHI. Now degeneracy is reflected by :

$$(5.1) \qquad 0 = t_1 = \dots = t_\ell < t_{\ell+1} \leq \dots \leq t_Q .$$

The maximum of w(t) is defined by index q - see (4.4) - for which

$$(5.2) \qquad \begin{aligned} r_q &= r_1 - \sum_{p=1}^{q-1} |\alpha_{i_p}| > 0 \\ r_{q+1} &= r_1 - \sum_{p=1}^{q} |\alpha_{i_p}| \leq 0 , \end{aligned}$$

taking into account that $r_1 = -d_j$, (5.2) can be rewritten as

$$(5.3) \qquad \begin{aligned} &\sum_{p=1}^{q-1} |\alpha_{i_p}| < -d_j \\ &\sum_{p=1}^{q} |\alpha_{i_p}| \geq -d_j . \end{aligned}$$

If for this q, it is true that $q > \ell$, then $t_q > 0$ as can be seen from (5.1) and despite of the presence of degeneracy a positive step can be made thus making a definite improvement in the measure of infeasibility. At the same time the number of infeasibilities may increase. This is however not considered an unfavourable situation because now - as a result of the transformation - some of the feasible zeroes may be replaced by infeasible non--zeroes thus decreasing the degree of degeneracy and promoting the subsequent non-degenerate iterations.

If $q \leq \ell$ then $t_q = 0$ and degeneracy prevents an increase in w(t) so the incoming variable enters the basis at zero. A simple case of this phenomenon was displayed in Fig. 7.

The above discussion can be summarised as:

Theorem 5.1. If $\sum_{p=1}^{\ell} |\alpha_{i_p}| < |d_j|$, where ℓ is as used in (5.1), then by changing the basis as defined in Theorem 4.2 the measure of infeasibility - despite of the presence of degeneracy - will be improved by

(5.4) $\quad D = w(t_q) - q(0) > 0$.

In any other case the progress will be zero.

It should be noted that while the number of infeasibilities may increase in certain cases the measure of infeasibility behaves monotonically while maximizing w(t) at each iteration. It is also true that the number of infeasibilities may drastically decrease in one iteration which is quite often the case though type-0 variables can usually be made feasible only one by one. In other words: there is no danger in allowing the free movement of the basic variables.

Computational experiences with DELPHI were especially favourable in the case of degenerate problems as is shown in Section 7.

6. Computational aspects

As it has already been explained the algorithm DELPHI determines the outgoing variable for a given incoming variable by maximizing w(t). Some additional features and requirements of this method are also to be noted.

6.1. Numerical stability

One of the reasons for occasional numerical troubles in the simplex method is the improper size of the pivot elements. Using the traditional pivot selection criterion this can hardly be overcome: if the minimum of the computed quotients is unique then there is no choice, if it is not unique then the quotient with a better-sized pivot element can be chosen.

Using DELPHI we are given a much greater flexibility in this respect. If the pivot corresponding to t_q is unacceptable because of its size then - usually losing optimality of $w(t)$ - we can go one step back if $q > 1$ and consider the pivot for $q := q - 1$. This procedure can be repeated if necessary until $q = 1$. On the other hand q can also be exceeded in the course of the search for a proper pivot element giving $q := q + 1$ until $q \leq Q$ and while $w(t_q) \geq w(t_0)$. In some cases it is worth losing optimality in $w(t)$ for the sake of finding "good" pivots.

6.2. Multiple pricing

Multiple pricing is a frequently used technique in LP packages for its economy in reducing the number of accesses to the background storage. Suboptimization is usually carried out using the principle of greatest change of the objective function.

The procedure DELPHI can easily be adjusted to this framework. Since the updated candidate columns are now available, DELPHI can be applied to each column. This determines not only the outgoing variable but also the progress - by Lemma 4.2 - that can be achieved enabling us to select the most favourable candidate.

6.3. Memory requirements

DELPHI requires the storage of the calculated T_{ℓ_i} and T_{u_i} values. The maximum number of these values is $2(m-1)$ because for each row, except for the objective row, at most both T_{ℓ_i} and T_{u_i} are defined. At the same time a permutation vector is also to be stored to register the correspondence between t_k-s of (4.1) and the T_{ℓ_i} and T_{u_i} values.

Arrays of this size are available in the so-called Π region which is active during pricing but inactive when DELPHI works.

6.4. Computational effort

Computing the T_{ℓ_i} and T_{u_i} values according to formulae (3.7) and (3.8) doesn't mean much extra computation since most of these quantities are computed when the traditional method is used. Computationally the only difference is that now these values are stored.

The next step is sorting the stored values as required by (4.1). The purpose of this step is to enable the direct performance of recursions (4.3) and (4.5) to determine the maximum of $w(t)$. Sorting all the Q items may be quite expensive if Q is large. Fortunately, it is not necessary to do that all the time. From observation usually only a few of the sorted t_i values are used to find the maximum of $w(t)$. Therefore it is best to use a sorting scheme which in step i gives a correct ordering for the first i items. Simultaneously with step i of the sorting, step i of the recursions (4.3) and (4.5) can also be computed and when the stopping rule (4.4) is satisfied the maximum of $w(t)$ is found and there is no need to sort the remainder of the t_i $(i = q+1,\dots,Q)$ values. H.J. Greenberg has proposed a one-pass scheme [4] for the reduction of the computational effort of finding the maximum of w(t). Both methods are only heuristic remedies since they don't give any saving when the worst case occurs (when we need all the Q items).

7. Experiences with DELPHI

DELPHI has been implemented in LP package LIPROS in such a way that the old subroutine PIVOT 1 has been replaced by DELPHI. (This was slightly disadvantageous for DELPHI because it was obliged to work with candidate columns that had been selected by an algorithm tuned for PIVOT 1.) LIPROS is an LP package for the R-10 computers produced in Hungary. These computers are small (main store is up to 64 Kbyte) but their processor is relatively fast. Therefore procedures which work mostly in memory and require little communication with the background storage are to be preferred. This is the case with DELPHI which doesn't require extra I/O operations but can save many of them by a sharp reduction in the number of iterations.

In the course of the comparative runs we found that the average time per iteration was practically the same for PIVOT 1 and DELPHI (the actual differences were within a 6% range). In Table 1 we present some of the run statistics which we found typical. The runs were carried out under identical circumstances (run parameters, etc.) for both PIVOT 1 and DELPHI. In each case the starting basis was the all-logical basis. The tableau shows the number of iterations in Phase I. The size of each problem is expressed by $m * n$, DOD = degree of degeneracy of the starting basis as a percentage.

Problem		PIVOT 1	DELPHI	Remark
N^{o}	size			
1.	62×70	43	34	25 equalities among constraints
2.	61×10	42	12	The all-logical basis was infeasible in every variable
3.	100×130	97	74	50 equalities among constraints
4.	170×120	unsolvable	278	60 upper bounded variables DOD = 96%
5.	237×184	not solved	107	55 upper bounded variables DOD = 99.5%

Table 1.

In Problem 2 all the constraints were of "≥" type. The measure of infeasibility rapidly improved while the number of infeasibilities increased a little in two cases,however in other cases it decreased drastically (as much as 10 - 20 infeasibilities were removed in one iteration). In general it seemed that the potential advantages of DELPHI were really effective in this example more than for the average problems.

Problem 4 was unsolvable with PIVOT 1 due to the unfavourable accumulation of round-off errors (with 4-byte floating point arithmetic) as was declared by LIPROS at about iteration 500. DELPHI also had troubles with this very degenerate problem (the phenomenon of Fig. 5) but at iteration 205 it could find a positive step forwards at the "expense" of increasing the number of infeasibilities from 1 to 5. After that it was a straightforward progress until the termination at step 278. Numerical inaccuracies didn't disturb the process (see 6.1 Numerical stability) and the result was accurate.

Problem 5 was solved only by DELPHI. It gave an interesting and characteristic experience. Even after applying a CRASH pass the new basis was highly degenerate. DELPHI could considerably reduce both the measure and the number of infeasibilities within few steps. Then came an "idleness": for 60 iterations only degenerate steps were made without any progress. At iteration 90 the number of infeasibilities suddenly soared from 1 to 97 while w(t) made a positive step ahead. The zero structure of the right--hand-side was practically destroyed and after that very efficient steps were made until iteration 107 when Phase I terminated.

8. ADACOMP, an adaptive composite Phase I procedure

DELPHI has proved to be a powerful tool for determining the outgoing variable in Phase I of the simplex method. Its power is first of all in the significant reduction of the number of iterations in Phase I. It has long been known that the proper selection of the incoming variable also heavily influences the total number of iterations required for obtaining a solution to an LP problem. There are several approaches which attempt to make good column selection: [1], [5] and others. These methods supply extra information at extra computational effort. One of the possible ways of reducing the total number of iterations is by considering the real objective function in Phase I. The purpose of this approach is that the first feasible point is as good as possible in the sense of optimality thus possibly reducing the number of steps required in Phase II. The usual way of doing this can be described as follows.

Let $z = c^T x$ be the true objective function which is to be maximized, and w be the negative of the sum of infeasibilities. Then the composite sum of

$$s = w + \lambda z \tag{8.1}$$

is formed with some $\lambda > 0$ value and s is maximized. This means that matrix A is priced for s. When no column can be selected the solution is s-optimal. If in the s-optimal solution w is zero then we are at an optimal solution of the original problem (the solution is z-optimal). If $w < 0$ then feasibility of the problem is still to be decided. To do this λ is set to zero and A is priced for w resulting in a pure Phase I procedure. If the problem has no feasible solution then an unlucky choice of λ and the effort to work with s of (8.1) as long as possible may delay the detection of infeasibility because there may be columns for which there is no improvement in w but due to the relative magnitudes of w, λ and z these columns will be candidates if we price A for s (since $s = w + \lambda z$). If the problem is feasible then in the course of the pure Phase I procedure the gain in the true objective function may be lost. Without further analysing the possible disadvantages of this procedure we simply refer to the fact that in the literature only very limited information is available on the success of this approach though most of the large LP systems are equipped with this composite facility.

We obtained quite favourable experiences with an adaptive composite procedure which has certain similarity with the method just described. However there are two points in which our method is different:

i) we allow a column to be a candidate if it shows improvement both for w and s of (8.1)

ii) we change the λ weight factor dynamically in accordance with the actual situation.

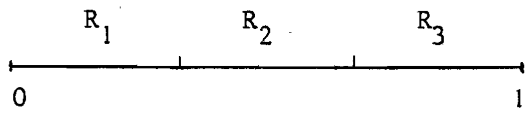
Let us denote by z_j the relative cost of column j if A is priced for z (the true objective function) and by d_j the relative cost if A is priced for w which is the same as defined in (3.4).

We describe the adaptive composite procedure - called ADACOMP - in a multiple pricing environment. Now N_s will denote the maximum number of columns to be selected in multiple pricing. At the beginning of the iterations λ will be given a non-negative initial value. During a major iteration (column selection) if there are more than N_s candidates with respect to w then a secondary selection rule is entered. This gives preference to those columns for which $d_j + \lambda z_j$ is more negative. The suboptimization can be performed using any of the known principles.

In the course of a major iteration two counters are used: L_1 counts the number of negative d_j values and L_2 counts the number of those negative d_j-s for which $d_j + \lambda z_j < 0$.

If $L_1 = 0$ then the problem is infeasible (no repricing is necessary).

For the next major iteration the value of λ will be defined in the following way. We use ρ to denote $\rho = L_2/L_1$. Clearly $0 \le \rho \le 1$ holds. The interval $[0,1]$ is divided into three parts (as shown below)



such that $[0,1] = R_1 \cup R_2 \cup R_3$ and the R_i-s are disjunct. The new value of λ is denoted by $\bar{\lambda}$ and is computed from

$$\text{(8.2)}\qquad \begin{array}{llll} \text{If} & \rho \in R_1 & \text{then} & \bar{\lambda} = g_1(\lambda) \\ \text{if} & \rho \in R_2 & \text{then} & \bar{\lambda} = g_2(\lambda) \equiv \lambda \\ \text{if} & \rho \in R_3 & \text{then} & \bar{\lambda} = g_3(\lambda)\ . \end{array}$$

Here $g_1(\lambda)$ and $g_3(\lambda)$ are functions for which $0 \le \bar{\lambda} = g_1(\lambda) < \lambda$ and $0 \le \lambda < \bar{\lambda} = g_3(\lambda)$ hold.

(8.2) can be interpreted so that when most of the candidates with respect to w are also candidates with respect to s ($\rho \in R_3$) then - in this favourable situation - movements in the direction of the true objective function can be given greater weight and this is achieved by increasing the value of λ.

In the opposite case optimality becomes less important than feasibility and this is expressed by decreasing λ. The second line of (8.2) means that the two components of s are well balanced and there is no need for changing λ.

It is easy to see that by (8.2) we created a great deal of flexibility. The actual choice of the parameters (sets R_1, R_2 and R_3 and functions g_1 and g_3) gives the opportunity of tuning the algorithm and adjusting it to the problem to be solved. The initial value of λ is also a run parameter.

It should be noted that ADACOMP requires some extra computations. In addition to d_j, z_j may also be needed; this can be computed by the inner product

(8.3) $$z_j = \Pi^T a_j$$

where Π is the simplex-multiplier of the true objective function. Π can be obtained from

(8.4) $$\Pi^T = e_\nu B^{-1} .$$

Here e_ν denotes that m dimensional unit vector which corresponds to the true objective function. In computer implementations (8.4) is usually computed by a BTRAN [9] operation. This can be done concurrently with the determination of the simplex multipliers of w as a BTRAN on two vectors thus requiring no extra pass of the eta-file. (The eta-file is the mathematical equivalent of B^{-1} and is usually stored on a secondary storage). The (8.3) inner product is only computed for those non-basic variables which require the composite evaluation.

9. Experiences with ADACOMP + DELPHI

In the LP package LIPROS we have combined DELPHI with ADACOMP. In this way all the favourable features of DELPHI can be effective. As was mentioned earlier LIPROS is not very sensitive to some extra computations.

DELPHI tends to sharply reduce the number of steps in Phase I. ADACOMP was expected to increase the number of iterations in Phase I but to considerably decrease the number of iterations in Phase II so that the total number of iterations and also the total computational effort would be decreased.

We made comparative run of LIPROS with DELPHI and LIPROS with ADACOMP + DELPHI. In addition, we solved one problem with different large LP packages.

Tests were carried out in a multiprogramming environment under different loading circumstances and therefore the CPU times do not express perfectly the total computational work of the LP solution but give a good indication of the tendencies.

During the test runs the free parameters of ADACOMP were fixed as follows:

- starting value for λ was set $\lambda = 0.5$
- $R_1 = [0,1/3)$; $R_2 = [1/3,2/3)$; $R_3 = [2/3,1]$
- $g_1(\lambda) = \lambda/2$ $\quad g_3(\lambda) = 2\lambda$.

In Table 2 we present the run statistics of 3 problems solved with 3 different values for N_s. The problems are identified by their sizes, row by column (m by n). IT denotes the total number of iterations, IT (PH-I) denotes the number of iterations in Phase I and CPU denotes the CPU time. The unusually large execution times are due to the computer. Its processor is fast relative to its communication with the background but not relative to the processors of other well-known small computers.

	$N_s = 4$		$N_s = 6$		$N_s = 8$	
	DELPHI	DELPHI + ADACOMP	DELPHI	DELPHI + ADACOMP	DELPHI	DELPHI + ADACOMP
Problem 41 × 60						
IT	76	64	79	54	66	37
IT (PH-I)	49	57	36	47	42	34
CPU	3'28.9"	3'03.8"	3'33.1"	2'16.3"	2'37.7"	1'37.5"
Problem 62 × 70						
IT	55	30	45	31	35	31
IT (PH-I)	34	29	37	29	31	29
CPU	1'33.8"	1'07.0"	1'09.6"	1'08.3"	0'59.8"	1'04.3"
Problem 100 × 130						
IT	136	83	145	84	125	88
IT (PH-I)	74	75	75	62	76	76
CPU	7'29.0"	3'36.9"	6'55.5"	4'12.1"	5'46.8"	4'15.9"

Table 2.

It is noteworthy that when ADACOMP was used only very few steps were left for Phase II. Other test runs with larger problems also support this observation, sometimes with much larger savings in the number of iterations. Since those problems were only occasionally solved we do not have much systematic experience with them. In fig. 8 we show a tipical behavior of the true objective function in Phase I.

We are not in the position of having easy access to different computers. However, with the kind assistance of colleagues at other institutes, problem 62 × 70 was solved with different packages under identical run parameters with $N_s = 4$. Since the computers were different we give only the total number of iterations in Table 3.

It would be incorrect to conclude that LIPROS is so powerful compared with other packages. We simply consider it a fortunate case. Still further experimentation would be necessary to be able to make a stronger statement in this respect.

10. <u>A generalization of DELPHI</u>

DELPHI is based on the relaxation of Condition 1.2. An even more general case can be obtained if both of the Conditions 1.1 and 1.2 are relaxed. This means that we allow not only the old basic variables but also the incoming variable to take an infeasible value.

The incoming variable can be infeasible in two ways:

- with a value greater than its upper bound,
- with a negative value while it is not a type-3 variable.

Type-3 variables are excluded from the further discussion because they enter the basis always at a feasible level.

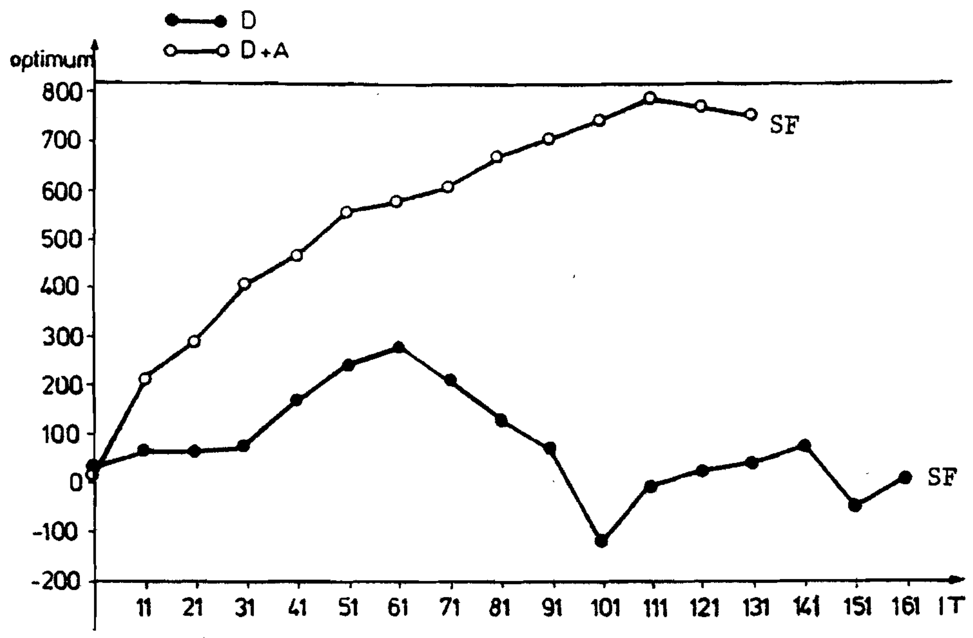


fig. 8

Movements of the true objective function in Phase I. D denotes the run with DELPHI, D + A is the run with DELPHI + ADACOMP.

At SF solution becomes feasible. (The first feasible solution of D + A is much closer to the optimum). Problem size is : 150 x 200.

LPS	(IBM/DOS)	81
MPS	(IBM/OS)	56
LIPROS	with DELPHI	55
LP-400	(ICL)	42
LIPROS	with DELPHI + ADACOMP	30

Table 3.

Let us suppose that a non-basic variable is at level zero and we want to change its value. Let us denote by $W_j(t)$ the corresponding infeasibility function in a similar sense as in Section 3. It can easily be seen that the measure of infeasibility has the following form now:

$$W_j(t) = \sum_{i \in I} (\beta_i - t\alpha_{ij})^- - \sum_{i \in I_u} (\beta_i - t\alpha_{ij} - u_i)^+ + G_j(t) \tag{10.1}$$

where

$$G_j(t) = \begin{cases} t & \text{if } t < 0 \\ u_j - t & \text{if } t > u_j \\ 0 & \text{otherwise .} \end{cases} \tag{10.2}$$

(10.1) can be written as

$$W_j(t) = w(t) + G_j(t) \ , \tag{10.1a}$$

where $w(t)$ is as defined in (3.6).

<u>Lemma 10.1</u>. Let us suppose that x_j is a non-basic variable and is at level zero. $W_j(t)$ can be improved by changing the value of x_j if

I. moving x_j in a positive direction, then

$$d_j = \sum_{i \in M} \alpha_{ij} - \sum_{i \in P} \alpha_{ij} < 0 \ , \tag{10.3}$$

II. moving x_j in a negative direction, then

$$d_j = \sum_{i \in M} \alpha_{ij} - \sum_{i \in P} \alpha_{ij} > 1 \tag{10.4}$$

holds.

Proof. In case I x_j moves in a positive direction that is $t > 0$ in (10.1). Since $G_j(t) = 0$ for $t > 0$ (and t small enough), $W_j(t)$ of (10.1a) reduces to $w(t)$ of (3.6) and so (10.3) to Lemma 3.1.

If x_j moves in a negative direction then the change of $W_j(t)$ is given by

(10.5) $$\Delta W_j = W_j(t) - W_j(0) = -td_j + \Delta G_j = -td_j + G_j(t) ,$$

since $G_j(0) = 0$. Taking into account that $G_j(t) = t$ if $t < 0$, (10.5) can be rewritten as

$$\Delta W_j = -td_j + t .$$

Here $d_j > 1$ is necessary for $\Delta W_j > 0$. This completes the proof. □

From (10.2) it is easy to verify that $G_j(t)$ is a concave function for $-\infty < t < +\infty$.

If x_j moves away from 0 then the values of the basic variables change according to (3.2). In the course of this their feasibility status can also change (namely when they cross the boundary of their feasibility range). The threshold values for t can be determined from (3.5). We investigate the two cases, when $t \geq 0$ and $t \leq 0$, separately.

I. $t \geq 0$.

The values of t at which the basic variables reach their respective boundaries are those defined by (3.7) and (3.8). In addition, there is one more t value, namely that one at which the incoming variable reaches its upper bound (if such exists):

$$T_u = u_j .$$

This defines a break point with $\alpha_i = 1$ (since $u_j = u_j/1$).

II. $t \leq 0$.

In this case the break points - as can easily be seen - are defined by

$$(10.6) \qquad \left.\begin{array}{ll} & T_{\ell_i} = \beta_i/\alpha_i < 0 \qquad (\alpha_i \neq 0) \\ \text{or} & T_{\ell_i} = 0, \ \text{if } \beta_i = 0 \wedge \alpha_i < 0 \end{array}\right\} i \in I$$

and

$$(10.7) \qquad \left.\begin{array}{ll} & T_{u_i} = (\beta_i - u_i)/\alpha_i < 0 \qquad (\alpha_i \neq 0) \\ \text{or} & T_{u_i} = 0, \ \text{if } \beta_i = u_i \wedge \alpha_i > 0 \end{array}\right\} i \in I_u .$$

Here also appears an additional break point: $T_\ell = 0$, with $\alpha_i = 1$, because of the lower bound of the incoming variable. Remarks made at (3.7) and (3.8) apply too.

The break points of case I and case II are sorted into the following order

$$(10.8) \qquad 0 \leq t_1 \leq \ldots \leq t_{Q_1}$$

and

$$(10.9) \qquad t_{Q_2} \leq \ldots \leq t_1 = 0 .$$

Now the analogue statement of Theorem 4.1 is the following:

<u>Theorem 10.1</u>. $W_j(t)$ is a piecewise linear concave function with break points (10.8) and (10.9). If we denote $r_1 = -d_j$ then the slope of the (k + 1)st interval of $W_j(t)$ is in case I

$$(10.10) \qquad r_{k+1} = r_k - |\alpha_{i_k}| \qquad k = 1,\ldots,Q_1 ,$$

and in case II

$$(10.11) \qquad r_{k+1} = r_k + |\alpha_{i_k}| \qquad k = 1,\ldots,Q_2 .$$

Proof. The proof of the theorem can be reconstructed by using arguments similar to those at Theorem 4.1. □

The shape of $W_j(t)$ is in case I similar to the basic patterns of $w(t)$ in Figures 5, 6 en 7, while in case II $W_j(t)$ is the reflection of $w(t)$ to the vertical axis with the remark that $t_1 = 0$ holds always.

The analogues of Lemma 4.1 and Lemma 4.2 can also be stated for finding the mximum of $W_j(t)$. Details of them are not given here.

Theorem 4.2 - without the remark concerning the type-1 iterations - is also valid here.

It may be instructive to see how the search for the maximum of $W_j(t)$ can go beyond the break point defined by the upper bound $t_k = u_j$ of the incoming variable. The k-th break point is preceded by the k-th linear interval with a slope r_k. The slope after this point is

(10.12) $\quad r_{k+1} = r_k - 1$

since $\alpha_{i_k} = 1$. If $r_{k+1} > 0$ then the (4.4) stopping rule is still not satisfied and the maximum of $W_j(t)$ is not achieved. From (10.12) $r_{k+1} > 0$ means that

(10.13) $\quad r_k > 1$.

In other words: if the slope of $W_j(t)$ till $t = u_j$ doesn't fall below 1, then the maximum of $W_j(t)$ is achieved at an infeasible value of the incoming variable.

It is worth seeing the analogy between (10.4) and (10.13), that is the conditions for the incoming variable to enter the basis at an infeasible level.

The Phase I procedure, based on the maximization of $W_j(t)$, is a generalization of the procedure DELPHI and is called DELPHI-A.

There are no computational experiences with DELPHI-A available since DELPHI-A has not been implemented yet.

Acknowledgement

The author wishes to thank A. Heppes for his attention and helpful suggestions throughout the preparation of this paper, and Susan Powell for improving the readibility of it.

References

[1] Benichou et al.: "The efficient solution of large-scale linear programming problems". Mathematical Programming Vol. 13 (1977) No. 3.

[2] Dantzig, G.B.: Linear programming and extensions. Princeton Univ. Press, 1963.

[3] Greenberg, H.J.: "Pivot selection tactics", in Greenberg, H.J. ed. Design and implementation of optimization software. Sijthoff and Nordhoff, 1978.

[4] Greenberg H.J.: Private communications. Pisa, July 1980.

[5] Harris, P.M.J.: "Pivot selection methods of the devex LP code", Mathematical Programming. Vol. 5 (1973) No. 1.

[6] Maros, I.: "Adaptivity in linear programming" (in Hungarian), Alkalmazott Mathematikai Lapok 2 (1976).

[7] Maros, I.: "A non-standard Phase I method", SZAMKI Tanulmanyok, Budapest, 1980/7.

[8] Maros, I.: "On determining the outgoing variable in Pase I of the simplex method" (in Hungarian). Alkalmazott Mathematikai Lapok, 6 (1980).

[9] Orchard-Hays, W.: Advanced linear programming computing techniques, McGraw, 1968.

[10] Wolfe, P.: "An extended composite algorithm for linear programming", The RAND Corporation, P-2373, 1961.

SUMMARY

A GENERAL PHASE-I METHOD IN LINEAR PROGRAMMING

I. Maros

Phase I of the simplex method finds a basic feasible solution to linear programming (LP) problems. Phase I procedures can also be used for generating feasible points of other problems with linear constraints, or even for checking the consistency of a system of linear equalities/inequalities.

The traditional Phase I methods work under the following conditions:

1.1 The incoming variable takes a feasible value.

1.2 The feasible basic variables remain feasible after a transformation (the number of infeasibilities does not increase).

1.3 The outgoing variable leaves the basis at a feasible level (lower or upper bound).

By relaxing condition 1.2 new possibilities arise though still within the frame of the simplex method. This paper presents a procedure based on this relaxation from which more efficient iterations can be expected in Phase I - especially in the presence of degeneracy - simply owing to the new way of determining the outgoing variable. This procedure - called DELPHI - is then combined with an adaptive composite column selection strategy - called ADACOMP - for determining the incoming variable. Though some extra compu-

tational effort is required this seems to be outweighed by the more favourable overall performance of the program based on the described method. This improved performance is partly due to the theoretically better numerical stability.

Some computational experiences are also reported.

In the last section DELPHI is further generalized by relaxing condition 1.1 too.