Author : P.A. Bruijs
E-mail : p.a.bruijs@wxs.nl
Affiliation : None, formerly Dr Neher Laboratories, Dutch PTT

Peter A. Bruijs

Looking for strong polynomiality in Linear Programming : Arguments, conjectures, experiments, findings and conclusion.

December 25, 2014

Abstract.
Until now it has been an open question whether the Linear Programming (LP) problem can be solved in strong polynomial time. The simplex algorithm with its combinatorial nature does not even offer a polynomial bound, whereas the complexity of the polynomial algorithms by Khachiyan and Karmarkar is based on the number of variables *n*, and quadratically on the length L of a bit-sequence representing the problem data.
The curious fact that non-linear algorithms would be needed to deliver the strongest complexity result for LP has served as the motivation for research producing the findings presented here.
In the arguments part this paper identifies characteristics of current LP-algorithms that hinder attaining strong polynomiality. This leads to conjectures concerning possible complexity outcomes on the basis of linear gradient-like algorithms, feasible gradient algorithms, that might offer a (strong) polynomial bound.
Experiments based on these algorithms, treating well-known problems from the literature, and a great many random problems have made clear that these algorithms generally do have strong characteristics, but their ultimate complexity depends on the precise implementation chosen.
Although an early conjecture of extremely strong polynomiality has been refuted by these experiments, a revised conjecture has not been refuted, but is firmly sustained by ample computational results, and by intuitive and ultimately theoretical arguments.

Keywords. linear programming - simplex algorithm - ellipsoid method - interior-point methods - gradient methods - computational complexity – strong polynomiality

Contents

1. Socratic introduction **3**

2. Characteristics of successful approaches to LP **5**
2.1 Simplex algorithm *5*
2.2 Polynomial algorithms: ellipsoid and interior point methods *5*

3. An appreciation **6**

4. The proposed gradient approach **7**

5. The initial conjecture on feasible gradient algorithms **11**
5.1 Specification of the initial version of the feasible gradient algorithm *11*
5.2 Additional specifications and further remarks *12*
5.3 Verification *14*
5.4 Falsification *14*

6. A revised conjecture on feasible gradient algorithms **15**
6.1 Specification of the revised version of the feasible gradient algorithm *15*
6.2 Ample additional computational experience *16*

7. Complexity analysis **17**

8. Conclusion **21**

9. Acknowledgement **21**

References **22**

Appendix Elaboration of a classical example **23**

1. Socratic introduction

The definition of the Linear Programming problem (LP) was established some 75 years ago. But despite the fact that an impressive body of optimality and duality theory regarding the problem has become available, and although several prolific algorithms are available to solve the problem in most cases, until recently there has been no algorithm solving the problem generally in strong polynomial time, meaning within a polynomial bound measured in the problem's dimensions: *m* and *n*, respectively the numbers of constraints and variables.
The first very successful attack to the problem, the simplex algorithm, has for some decades been the leading approach, and still has a very strong position, based on the clear and insightful approach it takes, and last but not least on the practical performance of advanced implementations of the algorithm.
However, scientific research has revealed weaknesses in special cases, where versions of the algorithm take an exponential number of iterations to reach the optimal solution, e.g. see the well-known Klee and Minty paper 'How good is the simplex algorithm ?' [3].
Since then the complexity of solving LP has become a prominent issue, and as such it did attract attention, resulting in new algorithms with their own strengths and weaknesses.
The extreme point approach of Dantzig's simplex algorithm was followed by completely different, essentially non-linear, approaches, taken by Khachiyan and Karmarkar.
However, whereas they present polynomial algorithms, the measure of complexity of their algorithms does not relate exclusively to *m* and *n*, and so it was still an interesting question whether a strong polynomial algorithm for LP would exist at all.

Since Dantzig 1950 [2], Khachiyan 1979 [4], and Karmarkar 1982 [5] a vast and impressive body of literature on the problem has been generated. Thereby practical as well as theoretical issues have been discussed extensively.
However directly after the publication of the results of Khachiyan and Karmarkar one might have been astonished, and I was, about the fact, that the linear problem at hand should derive its best complexity result based on non-linear algorithms. One might think for instance of Ockham's famous razor. It seemed natural in my view to conjecture that it would be possible to find a linear algorithm offering at least the same complexity result. This has been the starting point for my research project and the results are presented in this paper.

Looking at the extremely simple definition of the problem we see no more than a couple of purely linear constraints, in combination with the crucial element in the problem: the gradient of the objective, all elements that are absolutely constant throughout the problem space.
And it is not that hard to devise a simple algorithm that operates on the basis of the gradient.

As a first test I tried to solve the cycling problem of Beale with this algorithm, and there was no cycling whatsoever before reaching the optimum. In further tests, for example on some of the postal optimization problems I was working on, on assignment and transportation problems, and on relaxed travelling salesman problems, the approach worked very well too. Even the famous problem type defined on the perturbed Klee-Minty cube was solved without problems.

In hindsight these activities must be qualified as an effort of verification which would never have met the high scientific standard advocated by sir Karl Popper. For in the absence of a mathematical proof it will never be possible to attain more than a respectful conjecture on the

complexity of this kind of gradient approaches I tried. Furthermore, in the absence of a proof there should at least be an effort to refute the conjectures that were formulated.

M.J. Todd's paper "The many facets of linear programming" [6], published in 2002, gives an historic overview of the various algorithmic developments available at the time, and offers an excellent basis for the present analysis. Especially because at least one facet of the problem gets almost no attention.

Large parts of Todd's overview are concerned with mainstream algorithms, the simplex algorithm, and ellipsoid and interior-point methods.
'Gradient-like methods' are only covered in a tiny section 6.1 of his overview that states:

'These try to follow projected gradient directions from one feasible point to another. Since this seems like such a natural strategy, and apparently so obviously superior to edge-following methods, such methods have been proposed numerous times in slightly varying guises.'

On the basis of this statement, it is not quite clear in which way the gradient-like algorithms differ from the simplex algorithm, and from polynomial ellipsoid and interior point methods:

- Do not simplex and interior point methods follow a path of feasible points, and do they not use some gradient-related directions ?
- Why do they seem to offer 'such a natural strategy' ?
- What would make such an approach 'so obviously superior to edge-following' ?

Or are the qualifications in the above statement not at all adequate in the end ?

Furthermore and interestingly it may be observed that apparently no complexity result for these gradient-like methods can be given.

In this paper the questions raised above will be treated, leading to the presentation of linear, almost completely greedy, gradient-like algorithms, with characteristics strongly different from both the simplex algorithm, and from the ellipsoid and interior point methods.
The analysis of the complexity of these algorithms will lead to the assertion, that the right implementation delivers a strong polynomial algorithm for LP.

To arrive at this result we will look in more detail at the characteristics of currently used algorithms in order to detect the critical features determining their complexity. Next we give an appreciation of these characteristics and relate them to the characteristics of problem LP, and to general concepts used in optimization. By then the basic material is available for an informal introduction of the concepts and methods used in the alternative algorithms.
This is followed by a detailed description of the algorithms, supplemented with additional remarks concerning the characteristics of the algorithms, with details on computational aspects, with a completely worked out example - the cycling example once presented by Beale – and with ample computational experience from randomly generated test problems.

In the end it has become clear that an early version of the algorithm just operates quite efficient, but a modified version proves to ensure the desired complexity.

2. Characteristics of well-known successful approaches to LP

In this paragraph the aim is to look for some leading characteristics of the approaches taken by currently used LP-algorithms, thereby assuming basic knowledge on problem LP and related terminology.

2.1 Simplex algorithm

In a comic way it can be said that the simplex algorithm is basically based on basic solutions. And there is a clear argument to do so. For, as is well-known from the theory of linear optimization, an optimum of the problem can be found in a basic solution, which is an extreme point of the convex constraint set. Depending on the specific problem data one may encounter the situation that there are multiple optima, implying that there are not only several optimal basic solutions, but even that there are optimal non-basic solutions.
The algorithm essentially searches for the best basic solution in the set of all basic solutions. As extensively discussed in the literature, several pivot selection criteria can be used for the selection of an improved basic solution, each criterion having its own merits.
However, the sequence of intermediate basic solutions before reaching the optimum may be quite long as demonstrated by Klee and Minty in their paper “How good is the simplex algorithm ?” [3]. And in fact this leads to the conclusion that there are problems for which this sequence has an exponential length measured in terms of m and n, thereby refuting strong polynomial complexity of the simplex algorithm, independent of the pivot selection criterion chosen.
In practice, the expected performance of the algorithm proves to be quite good. But, although the optimal solution is a basic solution, in the light of the theoretical complexity it does not seem wise in every instance to constrain the solution sequence exclusively to basic solutions. And in principle there is no need to do so: the set of basic solutions is only a small subset of the complete convex space of feasible solutions to the problem.

2.2 Polynomial algorithms: ellipsoid and interior point methods

The approaches taken by ellipsoid and interior point methods are quite different in the sense that they originate from the realm of nonlinear convex optimization.
They share an interesting characteristic in the fact that they are based, as to the complexity result, on the assumption that the problem data can be represented by an L-bit sequence.
The complexity bound derived proves to be a function of the number of variables n and the length L of this sequence, the number of constraints m does not play an explicit part in this measure.

The ellipsoid algorithm is based on the fact that it is possible to construct from the problem data an ellipsoid enclosing all solutions, so including the optimum solution. In successive steps a sequence of shrinking ellipsoids is constructed. In every step a separating hyperplane can be constructed dividing the present ellipsoid in a part that contains the optimum and a part that does not. At a certain step it becomes possible to decide the optimum has been reached. In the process $O(n^4L)$ operations on $O(L)$-digit numbers are involved.

An interior point method tries to converge to the optimum from the inside of the feasible region. On the basis of an available feasible solution a transformation is made positioning this solution ‘in the centre’, meaning ‘with equal distance’ to the constraints, thereby providing room for improvement of the objective advancing through the inside of the feasible region.

In this case O($n^{3,5}L$) operations equally on O(L)-digit numbers are involved.

It is clear that essentially these methods are approximation methods with a sequence of enclosing ellipsoids, or explicit feasible non-basic solutions, both converging to the optimum.

3. An appreciation

The question now is: what can be derived from the characteristics described in the foregoing ? First of all, the analysis of the complexity of all approaches so far to solve LP does not offer a strong polynomial bound.

For the simplex algorithm what hinders is the fact that the algorithm reduces the space of candidate solutions to the set of basic solutions, the cardinality of which grows exponentially with the problem dimensions. And in this way it is possible to imagine, and it proves to be possible to construct, problems for which specific versions of the simplex algorithm generate solution sequences of super-polynomial length.

For the ellipsoid method the essential tool is a separating hyperplane, dividing the current ellipsoid into a ‘successful’ and an ‘unsuccessful’ part. However this hyperplane does not have a direct relation to the constraints defining the problem at hand.

For interior point methods the essential ingredient is the fact that intermediate solutions are ‘far from the constraints’, hopefully leading to far steps in the interior of the feasible region of the problem. So in this situation too there is no direct confrontation with the constraints defining the problem.

For the polynomial methods the available information on the problem is not fully exploited, the complexity measure remarkably being independent of m.

What does this mean in relation to the questions formulated earlier for gradient-like methods? A fully gradient-oriented method might be defined as a method that maximally moves in the direction of the gradient without violating any constraint.

What is special in the case of LP is, that the gradient is a unique and constant direction, and that there are only $m+n$ linear constraints defining a convex solution space.

Are these the crucial ingredients to be fully exploited ?
Does the following statement constitute a naïve ‘natural strategy’ as meant by Todd ?

> Follow the unique direction of the gradient, only and minimally deviating when a constraint would get violated, bearing in mind that there are only $m+n$ constraints for which a deviation may be necessary, and the fact that the solution space is convex.

Does the superiority to an ‘edge-following’ strategy in this case lie in the fact that no more than strictly necessary concessions are made in deviating from the gradient ?

After the publication of Khachian's result the concept of space dilatation has been mentioned as an important device in optimization, and furthermore in the realm of nonlinear convex optimization a saying is:

Separation = Optimization

One may wonder what separation device to use in constructing and selecting the optimum. And in the case of LP what would be a more natural choice than for the hyperplanes defined by the constraints to LP itself ? They in fact do cut off those parts of the problem space where the optimum cannot lie, and one needs the gradient direction starting from the current feasible solution as a guide to assess which hyperplanes should be used for separation.

An interesting, but clearly rhetoric, question, with the implicit suggestion about a possible number of separations needed to reach the optimum is:

How many separating hyperplanes are available anyway ?

Furthermore, in nonlinear optimization in every iteration and every intermediate solution point, we have to solve a direction-finding problem.
In the analysis of the complexity of their LP-algorithms, several authors do state this direction-finding problem explicitly as a quadratic optimization (sub)problem, with all consequences involved, as well for the computation of the optimum, as for the resulting complexity.
However in LP we are not confronted with a direction-finding problem, but on the contrary with a direction-following problem: the direction is there, we just have to follow in the best possible way. This suggests an approach of ´simultaneous coordinate descent´ in the direction of the gradient.
In the light of the foregoing it seems adequate to complement Klee and Minty's question "How good is the simplex algorithm ?" with the question "How bad is a gradient algorithm for LP ?"[7]
With the description of characteristics of available methods to solve LP and the appreciation added we have arrived at a point where the idea for an alternative method can be put forward.

4. The proposed gradient approach

Essentially the alternative approach proposed here to solve LP uses the information of the unique gradient to a maximum extent, bearing in mind that, starting from a feasible solution, one should not violate any of the constraints. And there are no more than these unique linear constraints that can hinder in proceeding in a direction following the gradient as close as possible, that is, with a maximal positive projection on the gradient.

An informal introduction to the alternative method can be given in the following way.

Let problem LP be defined in the standard primal form

$$\begin{array}{ll} \text{Max} & c^T x \\ \text{Sub} & A x \leq b \\ & x \geq 0 \end{array}$$

Where

$x = (x_1, \ldots, x_n)$
$c = (c_1, \ldots, c_n)$
A has row-vectors a_i , $i = 1, \ldots, m$
$a_i = (a_{i1}, \ldots, a_{in})$

all vectors in R^n ,

$b = (b_1, \ldots, b_m)$

a vector in R^m.

Let us assume now that we know an initial feasible solution $x(0)$.

(Such a solution, if it exists, can be computed gradient-based in an appropriate phase I procedure, whenever the algorithm for phase II starting from $x(0)$ proves to be successful.)

The first simple question to be asked seems to be how far the solution $x(0)$ can be improved into $x(1)$ simply, greedily, by proceeding for a step-size $s(1)$ in the direction of the gradient as specified by the problems object-vector c.

So

$$x(1) = x(0) + s(1)\, c$$

where $x(1)$ too has to be a feasible solution,
so,

$$a_i^{\,T}\, x(1) <= b_i \qquad \text{all } i$$

$$a_i^{\,T}\, x(0) + s(1)\, a_i^{\,T}\, c <= b_i \qquad \text{all } i$$

$$s(1) <= (b_i - a_i^{\,T}\, x(0)) / a_i^{\,T}\, c \qquad \text{all } i$$

so the maximum step without violating any constraint is

$$\min_i (b_i - a_i^{\,T}\, x(0)) / a_i^{\,T}\, c$$

But by then there is at least one constraint $a_i(imin1)$ that has to be considered binding,

$$a_i(imin1)^{\,T}\, x(1) = b_i(imin1)$$

so there is no more slack that can be used in the direction c of the original gradient.
In the meantime, it is clear that for the computation of an improving step one only has to consider those constraints for which one has in this first step

$$a_i^{\,T}\, c > 0$$

because a feasible solution is already available, so only these are candidates to become binding. In order not to violate the constraint $a_i(imin1)$ in further steps of our computation, one has to exclude every solution in the non-feasible half-space related to this binding constraint.
For the best direction that is left now, one has to exclude from the gradient c exactly and exclusively the component directing into this half-space.

The resulting feasible gradient $fg(1)$ in $x(1)$ may then be written as

$$fg(1) = c - \{ c^T a_i(imin1) / a_i(imin1)^T a_i(imin1) \} a_i(imin1)$$

and locally in $x(1)$ this is truly the best feasible direction.
In the next iteration in an analogous way one looks for the best value of

$$x(2) = x(1) + s(2)\, fg(1)$$

thereby determining a new binding constraint $a_i(imin2)$.
In further iterations one will have to exclude the non-admissible half-spaces related to $a_i(imin1)$ and $a_i(imin2)$.

It will prove to be convenient to determine an orthogonal basis consisting of vectors $aog(imin1)$, $aog(imin2)$, etc to span the combined non-admissible space related to binding constraints $a_i(imin1)$, $a_i(imin2)$, etc.

This may be done by means of the Gram-Schmidt procedure in the following way.

$$aog(imin1) = a_i(imin1) / a_i(imin1)^T a_i(imin1)$$

$$aog(imin2) = \{ a_i(imin2) / a_i(imin2)^T a_i(imin2) \} - \{ a_i(imin2)^T aog(imin1) / a_i(imin2)^T a_i(imin2) \} aog(imin1)$$

In this way one has

$$fg(1) = c - \{ c^T aog(imin1) \} aog(imin1)$$

$$fg(2) = c - \{ c^T aog(imin1) \} aog(imin1) - \{ c^T aog(imin2) \} aog(imin2)$$

while in step k+1 of the algorithm this will result in

$$fg(k) = c - \sum_{1 \le j \le k} \{ c^T aog(iminj) \} aog(iminj)$$

What is needed so far is the computation of the locally best direction to follow, by stripping the original gradient from the non-admissible directions derived from the binding constraints, and the computation of the feasible step-size in this direction.
Clearly the number of operations involved in these computations is only mildly polynomial related to m and n, the numbers of variables and constraints, while the operations used are just the simple arithmetic operations addition, subtraction, multiplication and division.
By now it becomes interesting to see how many iterations are needed, and also to see whether the optimum will be reached at all, for this determines whether or not a (strong) polynomial algorithm to solve LP would result in this way.

From the description given so far it is clear that every iteration is a greedy, but also forced, step, whereby one is not confronted with the choice for one or another pivot, or even one or another pivot selection rule.
But this situation only holds until iteration n is reached where it is inevitable that $fg = 0$!

The question then is how to diagnose this situation.

There seems to be no direction left for further optimization of the objective function.
However, optimization theory lends a hand by means of Farkas' Lemma [1], stating that if the optimum has been reached the following condition must be met :

c is in the cone of the normal vectors to the binding constraints,

or

$$c = \sum_i \mu(i)\, a_i \qquad \mu(i) \geq 0,\ a_i \text{ is a binding constraint}$$

As the set of binding constraints is known we are able to compute the multipliers $\mu(i)$ involved, and so to determine whether or not the optimum has been reached.
In the computation of the multipliers the orthogonal set of vectors *aog* derived from the binding constraints proves very useful.
The exact computation of the multipliers will be specified in the sequel.
Ultimately it may be concluded that the optimum has been reached, or else there is at least one constraint with a negative multiplier, indicating one or more constraints that locally should no longer be considered binding.
In the latter case we are able to re-compute the feasible gradient resulting in

$$fg > 0$$

and to move on following fg.

In principle the current basic solution may be taken as a fresh new starting point for the computation of the optimum, by declaring all constraints to be non-binding.
Because we are not yet in the optimum it will be possible to move on to a better solution, and as a consequence of the convexity of the feasible region the non-optimal solution found earlier will never be reached again, and informally spoken 'at least one of the constraints gets left behind' in this way. But this must happen then in every non-optimal basic solution that will be reached, hence the number of basic solutions until the optimum is reached is bounded in some way by the number of constraints in the problem.

Ultimately while the problem has a number $m+n$ of constraints we arrive at the conclusion that an algorithm based on ideas given in the foregoing should produce less than $(m+n)^2$ iterations, which, in combination with the mildly polynomial effort in the computations in an iteration, puts forward the perspective of strong polynomiality.

Because there may be several constraints with a negative multiplier we are confronted in an implementation with the question which single constraint or which group of constraints to free from their current binding status. This resembles by the way the situation in the simplex algorithm where we have to select one variable with a negative multiplier from the complete set of variables with a negative multiplier for the exchange with a current basic variable : do you choose the most negative multiplier or another one. In the simplex algorithm there is only room for one new single variable to enter the basis, while in the gradient concept presented here we may restrict ourselves to the removal of only one constraint from the binding set, or alternatively we might remove at most all constraints with a negative multiplier.

Hereby we have completed the description of our Euclidean approach to LP.
The presentation in the preceding material has been essentially complete but informal.

So apart from the definition of the problem as given earlier, a precise specification of an algorithm is needed now to be able to arrive at specific conclusions .

5. The initial conjecture on feasible gradient algorithms

Based on the arguments given, the first naïve conjecture on the complexity of gradient algorithms might be:

Conjecture 1: 'In every non-optimal basic solution of a feasible gradient algorithm
you lose one constraint
as a candidate for the set of binding constraints in the optimum,
namely the constraint with the most negative multiplier'.

This would mean that such an algorithm would have a maximum of $n+m$ iterations !
You might hope for the disappearance of the constraint with maximum negative multiplier, which would be a quasi-copy of the minimum relative cost rule in the simplex algorithm.
The corresponding version of such an algorithm is as follows.

5.1 Specification of the initial version of the feasible gradient algorithm

In case a feasible solution $x(0)$ is available one proceeds as follows :

(step 1) initialisation
set k=0, a feasible solution is $x(0)$, the feasible gradient is $fg(0) = c$ and identify index-sets $B(k)$ of binding, and $NB(k)$ of non-binding constraints, initially $B(k)$ empty, $NB(k) = \{ i \mid a_i \text{ not in } B(k) \}$

(step 2) compute the orthogonal set $BOG(k)$ for the binding constraints
by means of the Gram-Schmidt procedure, so

$$BOG(k) = \{ aog(i) \mid i,j \text{ in } B(k) ; aog(i)^T aog(j) = 0, j \neq i; aog(i)^T aog(i) > 0 \}$$

(step 3) compute the actual feasible gradient

$$fg(k) = c - \sum \{ aog(j) \text{ in } BOG(k) \} \{ c^T aog(j) \} aog(j)$$

(step 4) if $fg(k) > 0$: go to step 5, else

if $fg(k) = 0$: an optimality check must be performed,

so compute the multipliers on the basis of Farkas' Lemma
- if all multipliers > 0 : stop : the optimal LP solution has been reached
- if not, remove the constraint with most negative multiplier from $B(k)$, and go to step 2

(step 5) compute the maximum feasible step away from $x(k)$ along $fg(k)$

$$s(k) = \min_{\{ i \text{ in } NB(k) , a_i^T fg(k) > 0 \}} \{ b_i - a_i^T x(k) \} / a_i^T fg(k)$$

if $s(k)$ is infinite : stop : LP is unbounded

if the minimum is not unique, select a constraint i for which $a_i^T fg(k)$ is maximal and add the proper index in $B(k)$

(step 6) $x(k+1) = x(k) + s(k)\, fg(k)$

(step 7) set k = k+1 and go to step2

5.2 Additional specifications and further remarks

The computation of a feasible starting solution, if not available, can be done by the same algorithm by means of an appropriate object-vector based on the violated constraints starting from an arbitrary starting solution, analogous to phase I of the simplex algorithm.

If in step 5 the minimum for $s(k)$ is not unique, one will have to select as the new binding constraint the one *imin* for which $a(imin)^T fg$ has the greatest value. This is the constraint that is the 'most constraining' one in the set of newly found candidate binding constraints.

If in step 4 a constraint is removed from $B(k)$ because of a negative multiplier, the effect on fg is crucially different from the effect of the exchange of a basic and a non-basic variable in the simplex algorithm. Note that, contrary to the simplex algorithm, the feasible gradient algorithm in general does generate a path through the interior of a facet of the convex hull, thereby reaching the next generally non-basic solution.

In every iteration to solve the direction-finding, or rather direction-following, problem it is sufficient just to exclude from the current feasible gradient the direction into the half-space of a newly found binding constraint. There is no need to solve a quadratic programming sub-problem as suggested by several authors, the use of the binding constraints as separating hyperplanes suffices.

From the definition of the problem and the algorithm it is quite clear that the binding set cannot grow beyond n constraints. This means that for the feasible gradient algorithm degeneracy is virtually inexistent. The reason that in the simplex algorithm degeneracy may cause problems lies in the definition of the constraint qualification 'binding'. In simplex this is based on the fact that there is no slack, whereas in the feasible gradient algorithm it is based on the fact that the constraint hinders following the direction of the initial gradient c.

From the fact that in a basic solution x, where $fg = 0$, there is a unique representation of the basis in the index-set B, while furthermore in step 4 we will either conclude that x is optimal, or that there is at least one non-binding constraint in B meaning that we can 'revive' fg, and consequently will leave x in the next iteration, we must conclude that cycling will not occur.

Until an iteration is reached where $fg = 0$ every iteration is a greedy, but also forced, move, whereby one is not confronted with the choice for one or another pivot, or even one or another pivot selection rule !

The computation of the multipliers in step 4 in the test for optimality is derived from the fact that, according to Farkas' Lemma, in the optimum c may be written as a positive linear combination of the normal vectors to the binding constraints

$$c = \sum \{ i \text{ in } B(\text{step } 4) \} \mu(i)\, a_i \qquad \mu(i) > 0,\ i \text{ in } B(\text{step } 4)$$

or, as can be shown to be essentially equivalent

$$c = \sum \{ i \text{ in } BOG(\text{step } 4) \} \mu og(i)\, aog(i) \qquad \mu og(i) > 0,\ i \text{ in } BOG(\text{step } 4)$$

This is intuitively clear because we have to be sure that we do not deny any directions for use when excluding the space spanned by the normal vectors to the constraints in the binding set. But the same space is spanned by the orthogonal set *aog(i)* derived directly from the *a(i)* in *B*, their relation being defined by

$$aog(i) = a(i) / a(i)^{T} a(i) - \sum \{ j < i \} \{ a(i)^{T} aog(j) / a(i)^{T} a(i) \} aog(j)$$

However it should be clear that the *aog(i)* are constructed in a sequential manner, which implies that *aog(*1*)* equals *a(*1*)* , *aog(*2*)* is orthogonal relative to *aog(*1*)* , *aog(i)* is orthogonal relative to all *aog(j)* with $j<i$!
With this in mind the computation of the multipliers can be organized in the following way.

To start with we have

$$c = \sum \{ i \text{ in } BOG(\text{step } 4) \} \mu og(i)\, aog(i) \qquad (5.2.1)$$

with *il* being the last index entering *BOG* we get

$$c^{T} aog(il) = \sum \{ i \mid aog(i) \text{ in } BOG(\text{step } 4) \} \mu og(i)\, aog(i)^{T} aog(il)$$
$$= \mu og(il)\, aog(il)^{T} aog(il)$$

from orthogonality it follows that

$$aog(i)^{T} aog(i) > 0 \qquad \text{all } i$$
$$aog(i)^{T} aog(j) = 0 \qquad i \text{ not equal } j$$

So

$$\mu og(il) = c^{T} aog(il) / aog(il)^{T} aog(il)$$

So now that the multiplier for *il* is known we may write (5.2.1) as follows

$$c - \mu og(il)\, aog(il) = \sum \{ i \neq il \text{ in } BOG(\text{step } 4) \} \mu og(i)\, aog(i)$$

But this relation can be used in a quasi-recursive way to compute the other multipliers, because the left hand side is completely known.

5.3 Verification

In step 4 the algorithm needs an analysis to assert which of the constraints with a negative multiplier definitely are binding, and which are not. At first sight a plausible guess might be that the constraint with the most negative multiplier should be considered non-binding.
Since approximately 1985 an experimental code based on this assumption has been available to test the algorithm, and early verification efforts were successful. Starting with Beale's simple example for cycling of the simplex algorithm, several LP problems, e.g. postal planning problems, have been solved. All this in an old-fashioned mainframe environment.
Later a random problem generator has been built, to generate Transportation, Assignment, and (relaxed) Travelling Salesman Problems. More recently the algorithm has also been tested on the Klee-Minty type problems, and these problems too were solved successfully.
So Conjecture 1 seemed quite satisfactory. However, as stipulated before, in the absence of mathematical proof this is not sufficient evidence.

5.4 Falsification

Discussions with Kees Roos have resulted, only recently, in the development of a procedure for the generation of random sets of Linear Programs for pre-specified values of *m* and *n*.
In this manner a thorough effort of falsification of Conjecture 1 became possible.

And, although many test problems were needed, in the end the falsification succeeded, leaving us with the feeling that Conjecture 1 may be seen at least as an excellent heuristic .

So computational experience shows that the initial guess of Conjecture 1 is a wrong one, and even that it may lead to cycling, but in fact we don't have to guess about this question.

First of all, if at least one of the multipliers is negative we know from Farkas' Lemma that the present binding set is not the optimal one, so at least one of the constraints in this set will ultimately prove to be unbinding.
By further theoretical arguments we are, locally in a basic solution, absolutely sure that constraints with a negative multiplier are the candidate constraints 'to be left behind' as definitively non-binding.
Also, locally, the constraints with a positive multiplier definitely are locally essential, and therefore form a sure basis to compute a new version fg^* of the feasible gradient as a basis for further iterations. On the basis of our computational experience we come now with

Conjecture 2: 'In every non-optimal basic solution reached by a feasible gradient algorithm
you lose at least one constraint as a candidate
for the set of binding constraints in the optimum,
but it is impossible to identify which one of the constraints with
a negative multiplier will prove to be ultimately non-binding,
and as a consequence all constraints with a negative multiplier
have to be removed from the binding set, resulting
in a new tentative version fg^* of the local feasible gradient.

In case that there are more negative multipliers it is just the constraints involved that define the new search space locally left for improvement.
This leads to a second revised version of the feasible gradient algorithm.

6. A revised version of the feasible gradient algorithm

6.1 Specification of the revised algorithm

In case a feasible solution $x(0)$ is available one proceeds as follows :

(step 1) initialisation
set k=0, a feasible solution is $x(0)$, the feasible gradient is $fg(0) = c$ and identify index-sets $B(k)$ of binding, and $NB(k)$ of non-binding constraints, initially $B(k)$ empty, $NB(k) = \{ i \mid a_i \text{ not in } B(k) \}$

(step 2) compute the orthogonal set $BOG(k)$ for the binding constraints by means of the Gram-Schmidt procedure, so

$$BOG(k) = \{ aog(i) \mid i,j \text{ in } B(k) \; ; \; aog(i)^T aog(j) = 0, j \neq i; \; aog(i)^T aog(i) > 0 \}$$

(step 3) compute the actual feasible gradient

$$fg(k) = c - \sum \{ aog(j) \text{ in } BOG(k) \} \{ c^T aog(j) \} aog(j)$$

(step 4) if $fg(k) > 0$: go to step 5, else

if $fg(k) = 0$: an optimality check must be performed,

so compute the multipliers on the basis of Farkas' Lemma

- if all multipliers > 0 : stop : the optimal LP solution has been reached
- if not, remove the constraints with negative multiplier from $B(k)$, and go to step 2

(step 5) compute the maximum feasible step away from $x(k)$ along $fg(k)$

$$s(k) = \min_{\{ i \text{ in } NB(k) \, , \, a_i^T fg(k) > 0 \}} \{ b_i - a_i^T x(k) \} / a_i^T fg(k)$$

if $s(k)$ is infinite : stop : LP is unbounded

if the minimum is not unique, select a constraint i for which $a_i^T fg(k)$ is maximal

add the proper index in $B(k)$

(step 6) $x(k+1) = x(k) + s(k)\, fg(k)$

(step 7) set k = k+1 and go to step2

6.2 Ample additional computational experience

The actual performance of the revised version of the feasible gradient algorithm has been tested on a large number of random LP test problems, in the range from n=m=5 up to n=m=300, and additionally n=500,m=5 and n=5,m=500.
For n and m <= 100 there are 10.000 test problems in a series, for 150<=n and m<=300 there are 1000 test problems and for n or m=500 we have results for 100 problems.
Because we know we will lose at least one constraint as a candidate for the final i.e. optimal binding set with every non-optimal basic solution, it is interesting to look at the following indicator for the performance of the algorithm.

We compare the number of basic solutions in phase1 and 2 taken together needed to reach the optimum with the number of constraints in the problem.

In this fashion the worst case performance for a series concerning an (n,m) pair is defined in the following indicator:

(maximum number of basic solutions reached in phase1 + idem phase2) / (n+m)

The results in the table show a monotonous sub-linear growth for growing values of n+m !

		m								
		5	25	50	75	100	150	200	300	500
n	5	1,20	0,90	0,73	0,68	0,64	0,53	0,52	0,480	0,438
	25	0,07	0,56	0,65	0,56	0,51				
	50	0,04	0,21	0,42	0,38	0,45			0,254	
	75	0,03	0,13	0,22	0,27	0,29				
	100	0,02	0,08	0,23	0,17	0,28			0,200	
	150	0,013					0,20			
	200	0,010						0,16		
	300	0,007		0,006		0,005			0,100	
	500	0,004								

Apart from the more global characteristics of the experiments as given in the table above, there is more detailed information on the conduct of the revised algorithm.
Especially interesting is the analysis of the conduct of the signs of the values of the multipliers of the constraints for the diverse basic solutions passed during the computation.
Furthermore it is interesting to note the extremely diverse outcomes as to the disappearance from *B(*k*)* of the constraints with negative multiplier, even in small examples.
The following situations have been spotted in computational experiments:

- Indeed the constraint with most negative multiplier proves ultimately non-binding.
- The constraint with least negative multiplier proves ultimately non-binding.
- A subset of arbitrary cardinality, not identifiable on the basis of multiplier values, proves non-binding.
- All constraints with a negative multiplier are removed from the binding set, in a specific case even 10 constraints in a problem with n=20 and m=10.

7. Complexity analysis

After the presentation of the considerations leading to this research project, after the presentation of the informal idea for an alternative algorithm for LP, and the exact definition of the first and the revised version of that algorithm, and after the presentation of the results of series of test problems, we still have to assess the characteristics of the new algorithm in terms of its computational complexity.

First of all a general comment common to every numerical algorithm has to be made, namely that one needs a certain number of bits to store the problem data in order to be able to perform the necessary arithmetic operations correctly in an underlying Turing machine model.
This (always) leads to a factor *O(L)* in the complexity measure, where

$$L = \sum_{1\le i\le m} \sum_{1\le j\le n} (\log |a_{ij}| + 1) + \sum_{1\le i\le m} (\log |b_i| +1) + \sum_{1\le i\le n} (\log |c_i| +1) + \log mn + 1$$

Apart from that, we have to assess the complexity involved in the number of iterations that may have to be performed, as well as in the arithmetic operations per iteration.

Assuming the availability of a feasible starting solution *x(0)* the feasible gradient algorithm has two possible final outcomes :

- Either LP is unbounded as concluded in step 5,

- Or the LP optimum is found in step 4, by the verification of the sign of the multipliers, being all positive.

However, in both cases one might encounter the situation that one or more of the multipliers in step 4 proves to be negative, and this is the crucial case in the analysis of the complexity of the feasible gradient algorithm.
For if no negative multiplier is encountered, the feasible gradient algorithm generates a path toward the optimal solution in a straightforward way, by incrementing *B(*k*)* with as many constraints as needed to reach the optimum where *fg(*k*)*=0 , and clearly no more than *m+n* steps, or rather, even no more than *n* steps can be needed in that case.

However in the general case the complexity analysis, for Phase 2, has to consider three cases :

7.1 Consider first the situation that the conclusion that LP is unbounded is reached while in every iteration *fg(*k*)* > 0 . As there are no more than *m+n* constraints the number of iterations cannot possibly have grown beyond *m+n* before reaching this situation.
As the dimension of the problem space is *n*, this number will even not be greater than *n*.

7.2 Consider next the situation that all multipliers are positive the first time a solution is reached where *fg(*k*)* = 0 . As there are no more than *m+n* constraints the number of iterations cannot possibly have grown beyond *m+n* in this case too.
And as the dimension of the problem space is *n*, this number will be exactly equal to *n*.

7.3 Consider now the case that *fg(*k*)* = 0 and there is at least one negative multiplier.

The conclusion must be, that the optimum has not yet been reached, cf. Farkas' Lemma, and according to the (second version of the) algorithm we will now remove from $B(k)$ all constraints with a negative multiplier. But the question is then: what will happen next?

Except for cases in which it is discovered that LP is unbounded the following will happen:

We will see a sequence of basic solutions

$$BS(1),\ BS(2),\ \ldots .\ ,\ BS(\text{opt})$$

where because of step 4 the algorithm finds the binding sets

$$B(1) \neq B(2)\ ,\ B(2) \neq B(3)\ ,\ \ldots .\ ,\ B(\text{before opt}) \neq B(\text{opt})$$

respectively with solutions

$$x(B(1)),\ x(B(2)),\ \ldots .\ ,\ x(B(\text{opt}))$$

with

$$c^T x(B(1)) < c^T x(B(2)),\ \ldots .\ ,\ c^T x(B(\text{opt}))$$

Thereby it is clear, as already suggested above, that all successive binding sets differ in at least one constraint.
However, in this way we arrive at the following crucial question :

> Do the constraints with negative multiplier
> that are removed from $B(k)$ stay out of $B(l)$, for $l > k$, or
> could these constraints possibly re-enter $B(l)$ for some $l > k$,
> so in some subsequent basic solution ?

To answer this question we have to make use of the convexity of the feasible region :

First Convexity Lemma : There is no improved basic solution $x(l)$
with binding set $B(l)$ for a solution $x(l)$
with the same binding set $B(k)$ as for basic solution $x(k)$.

Proof
Suppose such an improved basic solution $x(l)$ would exist, then this would mean, that as well $x(l)$ as $x(k)$ are on binding constraints so they are in the convex hull of the feasible region of LP.
Because of the convexity of the feasible region we must conclude that every solution on the line connecting $x(k)$ and $x(l)$ is a feasible solution, and because the algorithm does not exclude any feasible solution from the set of possibly optimal solutions, there was no reason in $x(k)$ not to use this line to improve $x(k)$ to $x(l)$.
But this is in contradiction with the fact that in $x(k)$ the direction $x(l) - x(k)$ with a positive projection on c has not been identified as a feasible direction of improvement.

Additionally we have :

Second Convexity Lemma : At least one of the constraints in $B(k)$ with a negative multiplier has disappeared from $B(k)$ in all next improved basic solutions.

Proof
This follows directly from the First Convexity Lemma, for the same convexity argument as in the proof of the First Convexity Lemma goes for at least one of the individual constraints in $B(k)$ with a negative multiplier.

Third Convexity Lemma : Finding an improved basic solution $x(l)$ asks for disappearance of at least one and possibly more constraints from $B(k)$, thereby forcing the same number of non-binding constraint relative to $B(k)$ to enter $B(k+1)$.

Proof
According to the Second Convexity Lemma at least one constraint in $B(k)$ has to be discarded as a candidate for $B(opt)$, and moreover for every basic solution the binding set necessarily consists of n binding constraints.

The previous Lemma's lead to the next obvious conclusion :

Fourth Convexity Lemma : The maximum number of basic solutions the feasible gradient algorithm can find is m.

Proof
The Second Convexity Lemma asks for the disappearance of at least one constraint with a negative multiplier as a candidate for $B(opt)$, while the Third Convexity Lemma states the necessity of the entrance of at least one formerly non-binding constraint to replace this at least one constraint. In the first basic solution we have n constraints in $B(1)$, leaving m constraints that may replace constraints that are discarded as candidate for $B(opt)$.

The confusing fact in this line of reasoning may be, as it was for me, that we do not have to identify up-front which constraint it is, that will not be a member of $B(opt)$! Yet, as was to be expected intuitively and theoretically, our computational experiments do confirm this.

Lemma Iterations : The feasible gradient algorithm has an absolute upper bound of $n + m\,n$ iterations.

Proof
To reach the first basic solution n iterations are needed, and to reach the at most m next basic solutions no more than $m\,n$ iterations will be necessary.

Lemma Arithmetic Operations : The number of arithmetic operations per iteration of the feasible gradient algorithm is polynomially related to m and n.

Proof
From the definition of the algorithm in 5.1 we derive a maximum to the number of arithmetic operations for every step of the algorithm in every single iteration:

(step 2)	for the Gram-Schmidt procedure there are at most $2(n-1)+1$ inner products of n-vectors to compute the next orthogonal vector
(step 3)	at most n inner products of n-vectors to compute the next feasible gradient vector
(step 4)	at most $2n$ inner products of n-vectors to compute multipliers
(step 5)	at most $2(m+n)$ inner products of n-vectors to compute the maximum feasible step size
(step 6)	a multiplication on an n-vector to compute the next solution

whereby the inner product of *n*-vectors involves *n* multiplications, and whereby simple addition, subtraction, and comparison are of the same order.
So the effort per iteration is polynomial related to *m* and *n*, being *O((m+n)n)*.

The results presented in the preceding Lemma's logically lead to the main theorem :

Theorem Strong Polynomiality : The second version of the feasible gradient algorithm offers a strong polynomial bound for the solution of Linear Programming problems.

Proof
In the solution process every set of binding constraints *B(*opt*)* is generated by the process of successive selection of the strongest constraining constraint until :

- the binding set contains *n* constraints and a basic solution is reached, or
- the problem is found to be unbounded,

in step 4 resp. step 5 of the second version of the algorithm.
According to Lemma Arithmetic Operations and the fact that at most *n* iterations are needed to reach this situation the effort so far is clearly polynomial in *m* and *n* .
On reaching the first or any subsequent basic solution in step 4 with $fg = 0$, the computation of the multipliers corresponding with the constraints in the current binding set produces the information needed to decide whether the optimum has been reached (of course every basic solution may be succeeded by the conclusion that the problem is unbounded !)
The multipliers may turn out to be all positive or some may prove to be negative, however, the Fourth Convexity Lemma guarantees that no more than *m* basic solutions will be reached. Furthermore the Lemma Iterations guarantees that to reach these basic solutions no more than $n + n\,m$ iterations have to be performed.
Taken together with the resulting complexity in Lemma Arithmetic Operations results in a complexity for arithmetic operations and iterations of the second version of the feasible gradient algorithm of

$$O((n + n\,m)(m + n)n)$$

The complexity of the second version of the gradient algorithm is thereby

$$O((n3m + n2m2) L)$$

In addition to the specification of the complexity measure derived for the algorithm it is appropriate to remark that it may be called a highly pessimistic bound. The initial intuition as to the complexity has led to the idea that only $m+n$ iterations would be sufficient to reach the optimum. However, the fact that there is no decisive information in basic solutions to select the constraint(s) with negative multiplier that will not be present in the optimal binding set, results in the conclusion that possibly at most $n - 1$ constraints with a negative multiplier will have to be brought back into the binding set after every basic solution reached before the optimum.
This not only looks practically inconceivable, but all computational experience so far suggests a much milder practical bound, although especially for problems where $m \approx n$ and not very small, this effect might grow in an unpleasant fashion.
It may be recalled here that it has taken considerable effort to refute the first conjecture.

For a practical implementation of the feasible gradient idea it might be precisely the choice of how to treat negative multipliers, which offers plenty of room for heuristic experimentation.

8. Conclusion

In the conclusion of his review [5] mentioned earlier, Todd presents questions as to the expectations he had, as to future developments in the approach to LP.
It might be called remarkable to see that he did not have explicit hopes for developments in the area 'gradient-like methods', until now seemingly hardly researched for their complexity.

Here as a result of the analysis of current approaches to problem LP we have

- discussed the source of the non-polynomial character of the simplex algorithm, and concluded that a possibly unnecessary extra constraint has been introduced by the assumption that intermediate solutions should have to be basic solutions
- discussed available problem information not used in ellipsoid and interior point methods, and concluded that it may be unwise to avoid the direct confrontation with available information on the problem's constraints
- presented an alternative algorithm to solve LP that is explicitly based on the convexity of the feasible region and uses only purely linear concepts, as opposed to ellipsoid and interior-point methods, in line with LP's problem definition
- an algorithm which is in high conformity with Occam's razor as it only uses the concepts defining the problem, gradient and constraints, to solve the problem
- an algorithm leading to computations
    - that are clearly polynomial in effort per iteration
    - with a maximum number of iterations that is bounded by the fact that the convexity of the feasible region guarantees that in every basic solution reached by the algorithm at least one constraint is no longer a candidate to be present in the optimal binding set, meaning that the number of iterations is also clearly polynomially related to the number of variables and constraints
- so an algorithm leading to the LP-optimum as a strong polynomial alternative to the simplex algorithm, and to ellipsoid and interior point methods.

It has been G.B. Dantzig who remarked on several occasions[6] that a geometric view of the problem, clearly underlying the approach presented here, can be profoundly misleading!
And it was Todd who remarked in a personal communication that :

'The combinatorics and geometry of high-dimensional polyhedra is often counter-intuitive'.

These warnings were greatly appreciated, however, in the approach presented here the essential elements in problem LP are treated in a geometrically completely transparent way.
And as LP is a continuous optimization problem, treated here as such, one is absolutely not confronted with combinatorial issues.

The presentation of a theoretical view is the main purpose of the present paper.
The application of the algorithm to a classical problem of Beale serves as a detailed illustration, while additional computational results of the algorithm e.g. on several series of random LP test problems is presented in less detail.

9. Acknowledgement

Early discussions around 1980 with Erik van Doorn, as well as more recent discussions with Michael Todd, have been very helpful in the development of the result presented here. Lastly Kees Roos has been extremely helpful, especially in the refutation of the first conjecture.

References

1 J. Farkas. Ueber die Theorie der einfachen Ungleichungen.
J. Math, 124:1-24, 1902

2 G.B. Dantzig. *Linear Programming and extensions*.
Princeton University Press, Princeton, NJ, 1963

3 V. Klee and G.J. Minty. How good is the simplex algorithm ?
In O. Shisha, editor, *In equalities III*, 159-175, Academic Press, 1972

4 L.G. Khachiyan. A polynomial algorithm in linear programming.
Soviet Mathematics Doklady, 20:191-194, 1979

5 N.K. Karmarkar. A new polynomial-time algorithm for linear programming.
Combinatorica, 4:373-395, 1984

6 M.J. Todd. The many facets of linear programming.
Mathematical Programming, 91:417-436, 2002

7 P.A. Bruijs. How bad is a gradient algorithm for linear programming ?
Optimization_online, 2011

Appendix Elaboration of a classical example

To illustrate the operation of the feasible gradient algorithm we give the elaboration of Beale's classical example for cycling, by which we see a (weak) confirmation of the fact, named earlier, that the feasible gradient algorithm does not cycle.

The following Legend may be useful while reading the results of the algorithm:

fg the current feasible gradient vector

aifg inproduct of a(i) and fg

ineq inequality

nobi nonbinding

s_est step estimate on the basis of the current constraint

step current minimum of the step estimates (initially big)

in gs
number of constraints via Gram-Schmidt in the binding set

elements in a_g_s
number of nonzero elements in orthogonal representation of the binding set

big 'infinite'

void the constraint is irrelevant for the computation of the stepsize

iter iterationnumber

seqgs sequencenumber of entering the binding set

congs constraintnumber entering the binding set

multipliervalue
the value of the multiplier computed when fg = 0

analysis
gives an analysis of the multipliervalue : tobekept or con(straint)out(of B)

contyp
gives the constrainttype : eq(uality) or ineq(uality)

status gives the status of a constraint : bi(nding) or no(n)bi(nding)

rhs right hand side

```
>>>>
linear programming with the feasible gradient
result generated  date 15-9-2013 time 8:47:01

Beale's problem data  :  n = 4  m = 3  artif = 0

c : column,value     1  0.75000   2  -150.00000  3  0.02000  4  -6.00000
A: column,value
row 1 :   1   0.25000   2 -60.00000   3  -0.04000   4   9.00000
row 2 :   1   0.50000   2 -90.00000   3  -0.02000   4   3.00000
row 3 :   3   1.00000
b : row,value          1  0.00000  2  0.00000  3  1.00000

{step 1}
initially
z      0.00000
x      0.00000      0.00000   0.00000   0.00000
c      0.75000 -150.00000   0.02000  -6.00000
# elements in a_g_s totals     0
fg    0.75000  -150.00000   0.02000  -6.00000
{step 5}
  1 aifg   8946.18670 b    0.00 ineq  nobi  step         big   s_est     0.00000 imin   1
  2 aifg 13482.37460 b    0.00 ineq  nobi  step     0.00000 s_est     0.00000 imin   2
  3 aifg        0.02000 b    1.00 ineq  nobi  step     0.00000 s_est    50.00000 imin   2
  5 aifg    150.00000 b    0.00 ineq  nobi  step     0.00000 s_est     0.00000 imin   2
  7 aifg        6.00000 b    0.00 ineq  nobi  step     0.00000 s_est     0.00000 imin   2

ph2 iteration    1 # in gs    0 imin    2 step  0.00000

z        0.00000
x        0.00000   0.00000   0.00000   0.00000
slacks 0.00000   0.00000   1.00000
{step 2}
# elements in a_g_s totals     4
{step 3}
fg       -0.08130  -0.36672   0.05325 -10.98778
{step 5}
  3 aifg     0.05325 b    1.00 ineq  nobi  step          big   s_est    18.77869 imin   3
  4 aifg     0.08130 b    0.00 ineq  nobi  step    18.77869 s_est     0.00000 imin   4
  5 aifg     0.36672 b    0.00 ineq  nobi  step     0.00000 s_est     0.00000 imin   5
  7 aifg    10.98778 b    0.00 ineq  nobi  step     0.00000 s_est     0.00000 imin   7

ph2 iteration    2 # in gs    1 imin    7 step  0.00000

z        0.00000
x        0.00000   0.00000   0.00000   0.00000
slacks 0.00000   0.00000   1.00000
{step 2}
# elements in a_g_s totals     8
```

{step 3}
fg -0.08333 -0.00047 0.05333 0.00000
{step 5}
1 aifg 0.00552 b 0.00 ineq nobi step big s_est 0.00000 imin 1
3 aifg 0.05333 b 1.00 ineq nobi step 0.00000 s_est 18.75004 imin 1
4 aifg 0.08333 b 0.00 ineq nobi step 0.00000 s_est 0.00000 imin 4
5 aifg 0.00047 b 0.00 ineq nobi step 0.00000 s_est 0.00000 imin 4

ph2 iteration 3 # in gs 2 imin 4 step 0.00000

z 0.00000
x 0.00000 0.00000 0.00000 0.00000
slacks 0.00000 0.00000 1.00000
{step 2}
elements in a_g_s totals 11
{step 3}
fg 0.00000 -0.00001 0.05333 0.00000
{step 5}
3 aifg 0.05333 b 1.00 ineq nobi step big s_est 18.75000 imin 3
5 aifg 0.00001 b 0.00 ineq nobi step 18.75000 s_est 0.00000 imin 5

ph2 iteration 4 # in gs 3 imin 5 step 0.00000

z 0.00000
x 0.00000 0.00000 0.00000 0.00000
slacks 0.00000 0.00000 1.00000
{step 3}
fg 0.00000 0.00000 -0.00000 0.00000
{step 4, compute multipliers}
iter,seqgs,congs,multiplier,analysis,contyp,status,rhs
4 4 5 240.00000 tobekept ineq bi b= 0.00000
4 3 4 -1.25000 conout ineq bi b= 0.00000
4 2 7 3.00000 tobekept ineq bi b= 0.00000
4 1 2 -1.00000 conout ineq bi b= 0.00000
{step 4, remove constraints with negative multiplier}
7 4 5
7 5
{step 4, compute fg*}
elements in a_g_s totals 2
fg 0.75000 0.00000 0.02000 0.00000
{step 4, check for unjustified removals}
1 aifg 0.18670 b 0.00 ineq nobi step big s_est 0.00000 imin 1
2 aifg 0.37460 b 0.00 ineq nobi step 0.00000 s_est 0.00000 imin 2
3 aifg 0.02000 b 1.00 ineq nobi step 0.00000 s_est 50.00000 imin 2
{step 4, bring those back in the binding set}
ph2 iteration 5 # in gs 2 imin 2 step 0.00000
z 0.00000
x 0.00000 0.00000 0.00000 0.00000
slacks 0.00000 0.00000 1.00000
{step 2}

```
# elements in a_g_s totals     4
{step 3}
fg        0.00200   0.00000   0.04992   0.00000
{step 5}
  3 aifg    0.04992 b    1.00 ineq  nobi  step      big   s_est   20.03200 imin   3

ph2 iteration    6 # in gs    3 imin    3 step 20.03200

z         0.05000
x         0.04000   0.00000   1.00000   0.00000
slacks 0.03000   0.00000   0.00000
{step 3}
fg      -0.00000   0.00000   0.00000   0.00000
{step 4, compute multipliers}
 iter,seqgs,congs,multiplier,analysis,contyp,status,rhs
    6     4     3     0.05000 tobekept  ineq    bi   b=         0.00000
    6     3     2     1.50000 tobekept  ineq    bi   b=         0.00000
    6     2     7    10.50000 tobekept  ineq    bi   b=         0.00000
    6     1     5    15.00000 tobekept  ineq    bi   b=         0.00000
{step 4, optimum reached}

ph 2 no negative multiplier !

phase 2 best solution found :

z         0.05000
x         0.04000   0.00000   1.00000   0.00000
slacks 0.03000   0.00000   0.00000

# elements in a_g_s (seq_gs  congs  #elements *)
    1    5    1 *     2    7    1 *     3    2    2 *     4    3    2 *
# elements totals     6
# elements in a_ori  (constraintnr  #elements  *)
    1    4 *    2    4 *    3    1 *
# elements totals     9

constraints in optimal binding set
    2    3    5    7
constraint # times entering binding set  (constraintnr  # times entering *)
    1    0 *   2    2 *   3    1 *   4    1 *   5    1 *   6    0 *   7    1 *
total     6
constraint # times leaving binding set  (constraintnr  # times leaving *)
    1    0 *   2    1 *   3    0 *   4    1 *   5    0 *   6    0 *   7    0 *
total     2
(total entering – total leaving = 6 – 2 = 4 equals n=4 the dimension of the problem space)

iterations 6
(while n+m=4+3=7 !)
```