A conjugate Rosen's gradient projection method with global line search for piecewise linear optimization*

C. Beltran-Royo†

July 18, 2005

Abstract

The Kelley cutting plane method is one of the methods commonly used to optimize the dual function in the Lagrangian relaxation scheme. Usually the Kelley cutting plane method uses the simplex method as the optimization engine. It is well known that the simplex method leaves the current vertex, follows an ascending edge and stops at the nearest vertex. What would happen if one would continue the line search up to the best point instead? As a possible answer, we propose the *face* simplex method, which freely explores the polyhedral surface by following the Rosen's gradient projection combined with a *global* line search on the whole surface. Furthermore, to avoid the zig-zagging of the gradient projection, we propose a conjugate gradient version of the face simplex method. We have implemented this method in Matlab. This implementation clearly outperforms basic Matlab implementations of the simplex method. In the case of state-of-the-art simplex implementations in C, our Matlab implementation is only competitive for the case of many cutting planes.

Keywords: Linear programming, Kelley cutting plane method, simplex method, Rosen's gradient projection, conjugate gradient.

1 Introduction

The objective of this paper is to explore the computational performance of the *face simplex* (FS) method as a procedure to maximize an unconstrained *piecewise linear concave* (PLC) function, in the framework of the Kelley cutting plane method [14]. In the remaining of the paper cutting plane method will refer to the Kelley cutting plane method. Our purpose is not to compare the cutting plane method, to other methods used to maximize the Lagrangian dual function as for example: bundle methods [12], ACCPM (analytic center cutting plane method) [1], subgradient methods [21], etc. Instead, we are interested in comparing two optimization tools to be used within the cutting plane framework: the simplex and the FS method.

PLC functions are nondifferentiable and arise in the frame of the cutting plane method used for example to maximize the (Lagrangian) dual function [4]. The dual function is concave and

*This work was partially supported by Logilab, HEC, University of Geneva and the Spanish government, MCYT subsidy dpi2002-03330.

†cesar.beltran@hec.unige.ch, Logilab, HEC, University of Geneva, Switzerland.

very often not explicitly known. For this reason, first order information of the dual function (cutting planes) is gathered by means of black-box procedures also called oracles. The cumulated set of cutting planes constitutes a PLC outer approximation to the unknown dual function. At each cutting plane iteration one maximizes and improves this PLC approximation to the dual function.

The FS method is an alternative to the simplex method to maximize a PLC function. It has two inherent advantages: First, the FS method does not need the expensive initialization of the simplex method (to find an initial vertex). Second, in the case of a PLC unconstrained function, it may happen that there is no optimal vertex. In contrast with the FS method, the simplex method simply will either not be able to optimize such a function or will need to bound the variables in an artificial way, if possible.

It is well known that the simplex method at the current vertex takes an ascending edge and stops at the other vertex of the edge. *Question I:* What would happen if instead of stopping the line search at the second vertex, one continues the line search on the whole PLC surface up to the best point? Firstly, we would obtain a point at least as good as the one given by the simplex method in terms of objective value. Secondly, most probably the next iterate would not be a vertex any more and therefore the ascending edge direction proposed by the simplex method would not exist at this point. In this situation, the FS method proposes to follow the Rosen's gradient projection direction [19].

However, while the Rosen's gradient projection method performs a *local* line search (within the current face of the associated polyhedron), the FS method performs a *global* line search on the whole polyhedral surface. Both simplex and FS method use the Rosen's gradient projection as the feasible direction. They differ on the line search. In fact, the simplex method has the same behavior as the Rosen's gradient projection method started at a vertex (except for streamlining the linear algebra) [4].

Numerous gradient-like methods have been proposed to solve linear programs. In fact they appeared at the same time as the simplex method [5]. Later references are the Zoutendijk's method of feasible directions [25], the already mentioned Rosen's projected gradient method and the constrained gradient method [15], among others. More recently the steepest descent gravitational method has been proposed [6].

Two main aspects determine a gradient projection method. The direction search and the line search. Regarding the direction search, we also encounter two principal approaches: projection of the gradient on the entire polyhedral surface, not to be mistaken with the projection of the gradient on the current face (here we are assuming than all the problem constraints are linear). We will use the terms *gradient projection* for the first case and *Rosen's gradient projection* for the second one. Regarding the line search, we also encounter two principal approaches: line search on the current face and line search on the whole polyhedral surface. We will use the terms *local line search* and *global line search*, respectively.

The gradient projection method has guaranteed convergence for both local and global line search. But, it requires the solving of a demanding quadratic programming problem in order to project the gradient on the constraint surface. Closely related to the gradient projection, there is the steepest ascent method [12], which also requires to solve a quadratic programming problem at each iteration. It has guaranteed convergence only for local line search. If one wishes to perform long steps (global line search) along an *approximate* steepest ascent, one has to use an *outer approximate subdifferential* instead of the pure subdifferential [18].

In contrast, the computation of the Rosen's gradient projection is far less demanding since it only requires the product of the gradient by a projection matrix [4]. In this case we have the steepest ascent *on the current face*. In [7], the Rosen's gradient projection is used in combination with local line search to solve the uncapacitated facility location problem. The global line search combined with the gradient projection has been used in [10] to solve the graph partitioning problem. To our knowledge the combination of the Rosen's gradient projection with a global line search has never been used. The FS method fills this gap and constitutes a possible answer to above Question I.

A second issue addressed in this paper is the zig-zagging inherent to gradient like methods. In the differentiable case the zig-zagging is addressed by the conjugate gradient or equivalently, by the partan method [16]. Here we adapt to the case of a PLC function, the partan method. In our nondifferentiable problem the partan method is used only as a strategy to deflect the projected gradient, in contrast with the differentiable case where some conjugacy requirement is enforced. Conjugate subgradient methods have already been used to *approximately* solve nondifferentiable problems [23, 8]. Here, the partan FS method will compute an *exact* optimum of the PLC function.

This paper is organized as follows. In section 2 we state the PLC problem and associated notation. In section 3 the FS method is introduced. The global line search by the radar method is presented in section 4. A short discussion on the convergence of the FS method is presented in section 5. The partan FS method is introduced in section 6. Finally the numerical tests and conclusions can be found in sections 7 and 8, respectively.

2 Statement of the problem and notation

In the cutting plane method we generate a set of hyperplanes (cuts) $\{\pi_j \equiv z = s'_j y + b_j\}_{j \in J}$ with $J = \{1, \ldots, m\}$ whose lower envelope describes a piecewise linear concave (PLC) function:

$$F(y) = \min\{F_j(y) \mid j \in J\},$$

where $F_j(y) = s'_j y + b_j$.

At each iteration of the cutting plane method one maximizes F (we call it the *PLC problem*):

$$\max_{y \in \mathbb{R}^{n-1}} F(y). \tag{1}$$

This problem can be rewritten as:

$$z^* = \max_{(y,z) \in \mathbb{R}^n} \{z \mid z \leq s'_j y + b_j,\ j \in J\}. \tag{2}$$

By defining $x = (y, z) \in \mathbb{R}^{n-1} \times \mathbb{R}^1$, $a_j = (-s_j, 1)$ and $G(x) = z$, the PLC problem is equivalent to:

$$\begin{array}{l} \max_{x \in \mathbb{R}^n}\{G(x) \mid a'_j x \leq b_j,\ j \in J\} = \\ \max_{x \in \mathbb{R}^n}\{G(x) \mid Ax \leq b\}. \end{array} \tag{3}$$

Therefore the PLC can be seen as the unrestricted maximization of the (nonsmooth) PLC function $F(y)$ in the y-space (formulation (1)) or as the constrained maximization of the (smooth) linear function $G(x)$ in the enhanced x-space (formulation (3)).

The notation is as follows:

α^k denotes the kth term of the sequence $\{\alpha^k\}$,

$(\alpha)^k$ to write a power we use parenthesis,

x_i ith component of vector x,

$x = (y, z) \in \mathbb{R}^{n-1} \times \mathbb{R}^1$, partition of x into *ground variables* y and *epigraph variable* z,

$\text{Proj}_E(\cdot)$ Orthogonal projection onto the space E,

$\text{Proj}_y(\cdot)$ Orthogonal projection onto the first $n-1$ coordinate space, e.g., $\text{Proj}_y(x) = y$,

$g^k = \nabla G(x^k)$,

$e_n = (0, \ldots, 0, 1)$ nth canonical vector of $\mathbb{R}^n$,

$J(x) = \{j \in J \mid a_j' x = b_j\}$ active index-set associated to the active constraints at x,

$J^k = J(x^k)$,

$J^k \setminus h = J^k \setminus \{h\}$ set difference,

A_J sub matrix of A with rows indexed by J,

$A^k = A_{J^k}$,

$b^k = b_{J^k}$,

$S^k = \{x \in \mathbb{R}^n \mid A^k x = b^k\}$ active linear manifold at x^k,

$E^k = \{x \in \mathbb{R}^n \mid A^k x = 0\}$ vector space associated to S^k,

$S_y^k = \text{Proj}_y(S^k)$,

$E_y^k = \text{Proj}_y(E^k)$,

$X = \{x \in \mathbb{R}^n \mid Ax \leq b\}$,

FS Face simplex,

PLC Piecewise linear concave.

Assumption 1: $\{a_j\}_{j \in J(x)}$ is a set of independent vectors for any $x \in \mathbb{R}^n$.

Assumption 2: The PLC problem is bounded ($z^* < \infty$).

3 The face simplex method

The face simplex (FS) method is intended to solve the PLC problem (3) and 'only' differs from the Rosen's gradient projection method in the scope of the line search (see Remark 1 c)). The FS method can be summarized as follows:

Face simplex method:

Step 0. *Initialization:* Take $x^0 \in X$ such that $J(x^0) \neq \emptyset$, $\epsilon > 0$ and set $k = 0$.

Step 1. *Compute the Rosen's gradient projection:*

$$\begin{aligned} J^k &= J(x^k), \\ P^k &= I - A^{k\prime}(A^k A^{k\prime})^{-1} A^k, \\ u_j^k &= (A^k A^{k\prime})^{-1} A^k g^k, \quad j \in J^k, \\ u_h^k &= \min\{u_j^k \mid j \in J^k\}. \end{aligned}$$

If $\|P^k g^k\| < \epsilon$ and $u_h^k \geq 0$ then stop (in this case x^k is a Karush-Kunhn-Tucker point), else, choose

$$d^k = \begin{cases} P^k g^k & \text{if} \quad \|P^k g^k\| > u_h^k \epsilon, \\ P_{J^k \setminus h} g^k & \text{if} \quad \|P^k g^k\| \leq u_h^k \epsilon, \end{cases} \tag{4}$$

where $P_{J^k \setminus h} = I - A'_{J^k \setminus h}(A_{J^k \setminus h} A'_{J^k \setminus h})^{-1} A_{J^k \setminus h}$.

Step 2. *Global line search along $d_y^k = Proj_y(d^k)$:*

$$\begin{aligned} x^{k+1} = (y^{k+1}, z^{k+1}) = \text{arg max} \quad & z \\ \text{s.t.} \quad & (y, z) \in X \cap H^k, \end{aligned} \tag{5}$$

where

$$H^k = x^k + \text{span}\{d_y^k, e_n\}.$$

Set $k = k + 1$ and go back to Step 1.

Remark 1 *a) Note that $g^k = (0, \ldots, 0, 1) = e_n$ for all $x \in \mathbb{R}^n$. Then in the previous algorithm $g^k = e_n$ for all k.*

b) The FS method corresponds to the gradient projection method of Rosen with an enlarged linear manifold at Step 2. That is, in the Rosen's gradient projection method, the line search is restricted to the one dimensional linear manifold $\widetilde{H}^k = x^k + span\{d^k\}$ instead of H^k, which is two dimensional.

c) The Rosen's gradient projection d^k corresponds to the steepest ascend direction within F^k, the face associated to x^k in the polyhedron X (except when F^k is a vertex of X). In a related method, the steepest ascend method, one computes the (global) steepest ascend direction at x^k, which may be different form d^k.

4 Global line search by the radar method

In this section we focus on the solution of the global line search (5). This problem is equivalent to maximizing a one dimensional PLC function $f(\alpha)$ since the aim of (5) is to find a highest point within a vertical two dimensional slice of the polyhedron X. Furthermore $f(\alpha)$ is the lower envelope of the lines $\{r_j(\alpha)\}_{j\in J}$, where $r_j(\alpha) = m_j\alpha + n_j$ is the equation of the intersection line of hyperplane π_j with the two dimensional linear manifold H^k. That is

$$f(\alpha) = \min\{r_j(\alpha) \mid j \in J\}.$$

The expression for m_j and n_j is given in the following lemma.

Lemma 1 *If $\pi_j \equiv z = s'_j y + b_j$ and $H^k = x^k + span\{d^k_y, e_n\}$, with $x^k = (y^k, z^k) \in \mathbb{R}^{n-1} \times \mathbb{R}$, then $\pi_j \cap H^k$ defines a line whose equation is:*

$$r_j(\alpha) = m_j\alpha + n_j$$

with $m_j = s'_j d^k_y$ and $n_j = s'_j y^k + b_j$.

Proof: Let $\{(y(\alpha), z(\alpha)) \mid \alpha \in \mathbb{R}\}$ be the line $\pi_j \cap H^k$. Since $\pi_j \equiv z = s'_j y + b_j$ and $y(\alpha) = y^k + \alpha d^k_y$, then

$$z(\alpha) = s'_j y(\alpha) + b_j = \alpha s'_j d^k_y + s'_j y^k + b_j.$$

The result follows from the fact that $r_j(\alpha)$ is equivalent to $z(\alpha)$. ∎

The global line search (5) can be then reformulated as:

$$\alpha^* = \arg\max_{\alpha\in\mathbb{R}} f(\alpha) \tag{6}$$

Without loss of generality, we assume that α^* is the smallest optimal point in case of multiple optima. Furthermore, f is parameterized such that $f(0) = 0$ and $\alpha^* > 0$. The graph of f in the interval of interest $I = [0, \alpha^*]$ is the union of r line segments that join the breaking points $p_i = (a_i, b_i) \in \mathbb{R}^2$, $i = 1, \ldots, r$. Furthermore, $p_0 = (0, 0)$ and $p_r = (\alpha^*, f(\alpha^*))$. The interval I is partitioned into

$$I_1 \cup I_2 \cup \ldots \cup I_r = [a_0, a_1[\cup [a_1, a_2[\cup \ldots \cup [a_{r-1}, a_r].$$

The point a_r is characterized as the only point where the left derivative of f is strictly positive and the right derivative is negative, i.e., $f^-(a_r) > 0$ and $f^+(a_r) \leq 0$. This property could be used to solve the global line search: Start at $a_0 = 0$, then iterate from breaking point to breaking point up to the first breaking point with a change of sign in the lateral derivatives (*next breaking point method*).

The number of iterations in the next breaking point method equals the number of breaking points. To avoid the potentially large number of iterations of the next breaking point method, we use an improved version of the *radar method*. Before of introducing the radar method we need to partition the set of lines that define f. We consider *active lines* in $[0, +\infty[$ indexed by J_a, that is, lines that intersect with the graph of f in more than one point within the interval $[0, +\infty[$.

Analogously, we consider *inactive lines* indexed by J_i. We also consider *strictly positive lines* indexed by J^+, that is, lines with strictly positive slope. Analogously, *negative lines* indexed by J^- (negative lines may have null slope). By crossing these two attributes, we partition J into four sets :

$$J = J^+ \cup J^- = J_a^+ \cup J_i^+ \cup J_a^- \cup J_i^-.$$

Since in $[0, \alpha^*]$ we have r breaking points, we will have r segments $[p_i, p_{i+1}[$ $(i = 0, \ldots, r-1)$. Also, considering that these segments form the graph of a concave function, their slopes will form a decreasing sequence: $m_1 > m_2 > \ldots > m_r > 0$.

At iteration k, the radar method approximates $f(\alpha)$ by only two of its defining lines: $r^k(\alpha)$, and $s^k(\alpha)$: r^k is the active line at α^k (the one with the lowest slope if α^k corresponds to a breaking point) and s^k is the *stopping line*, i.e., the first negative line that intersects with r^k as α increases . The intersection point of r^k and s^k is the point $(\alpha^{k+1}, r(\alpha^{k+1}))$, which gives the next radar iterate α^{k+1} and an upper bound to the optimal value ($r(\alpha^{k+1}) \geq f(\alpha^*)$). See [2] for more details. The radar method can be summarized as follows:

Radar method:

Step 1. *Initialization:* Take $\alpha^0 = 0$ and set $k = 0$.

Step 2. *Compute next radar iterate:*

$$\alpha^{k+1} = \min \left\{ -\frac{m^k - m_j}{n^k - n_j} \mid j \in J^- \right\}. \tag{7}$$

Step 3. *Stopping criterion:* If $\alpha^{k+1} = \alpha^k$, then stop, since α^{k+1} maximizes $f(\alpha)$. Else, set $k = k + 1$ and go back to Step 1.

Remark 2 *a) A single iteration version of the radar method was first used in [2], thus obtaining a suboptimal solution of the global line search. In this paper, the aim of the radar method is to compute an optimal solution of the global line search.*

b) In [3] it is seen that in order to obtain an ϵ-optimum for the the global line search, i.e., an α^ϵ such that $|\alpha^\epsilon - \alpha^*| < \epsilon$, the number of radar iterations does not depend on the number of breaking points, but on the ratio between the smallest and largest slopes of the positive active lines. That is, for all k:

$$|\alpha^k - \alpha^*| < \left(1 - \frac{m_r}{m_1}\right)^k |\alpha^0 - \alpha^*|'. \tag{8}$$

5 Convergence proof in $\mathbb{R}^3$

The Rosen's gradient projection method for linear constraints was first published in 1960 without a convergence proof. The mathematical programming community had to wait until 1985 to see the first convergence proof but only restricted to dimension 3 [24]. Finally, in 1989 Du and Zhang [9] gave a general convergence proof.

Similarly, for the FS method we have not yet found a general convergence proof but only a convergence proof restricted to dimension 3. This simple result encouraged us to keep searching

for a general convergence proof and to empirically test the FS method. We wish to point out, that in contrast with the FS method, the steepest ascent method combined with global line search may fail to converge to a KKT point even for dimension 3 when maximizing a PLC function, [13].

The following lemma will be used in the proof of the convergence theorem (see [3] for a proof).

Lemma 2 *(radar method)*

a) If $\alpha^k \in [a_i, a_{i+1}[$ *then* $\alpha^{k+1} \in [a_{i+1}, \alpha^*]$.

b) $f(\alpha^{k+1}) > f(\alpha^k)$ *for* $k = 0, 1, \ldots$

c) $G(x^{k+1}) = f(\alpha^*) > f(0) = G(x^k)$ *for* $k = 0, 1, \ldots$

Theorem 1 *The FS method applied to solve the PLC problem in* $\mathbb{R}^3$ *converges to an optimal point or detects unboundness of* $G(x)$ *after a finite number of iterations.*

Proof: The proof is similar to the convergence proof of the simplex method. Now instead of exploring vertices, the FS method explores vertices and edges. If the polyhedral set X is included in $\mathbb{R}^3$ then its faces are points, edges or two-dimensional faces. In the radar method, x^{k+1} is defined by the intersection of two or more lines which correspond to the intersection of two or more hyperplanes. This means that x^{k+1} is a vertex or lays on an edge. By Lemma 2 c) vertices can only be visited once. Again, by Lemma 2 a)-b) an edge can only be visited once. Given that the number of vertices and edges is finite, the FS method either will attain a maximum after a finite number of iterations or will encounter an unbounded edge. ■

6 Avoiding the zig-zagging

Since the FS method is a gradient type method, it can suffer from zig-zagging specially when the graph of F has an 'elongated' form. To overcome the zig-zagging, one can use the partan (parallel tangents) method [20], a simple method that is equivalent to the conjugate gradient method in the case of a quadratic function. In this section we adapt the partan method to the case of a PLC function. Note that, in our non differentiable problem, the partan method is only used as a strategy to deflect the projected gradient, in contrast with the differentiable case where some conjugacy requirement is enforced.

The pure partan method combined with the FS method is as follows. Note that we work on the ground space variables $y \in \mathbb{R}^{n-1}$. Start at an arbitrary y^0 and compute y^1 by the FS method. Then at each iteration $k \geq 1$ two steps are performed: First, from the point y^k, compute the intermediate point $y^{k+\frac{1}{2}}$ by the FS method. Second, from the point $y^{k+\frac{1}{2}}$, compute y^{k+1} as the best point in the line $L = y^{k+\frac{1}{2}} + \text{span}\{y^{k+\frac{1}{2}} - y^{k-1}\}$. This double step process continues for $n-1$ iterations, and then a new partan cycle is restarted with a simple FS step.

Definition 1 *Let us define the interval* $I(\delta) =]-\delta, \delta[$. *Given a function* $F(y) : \mathbb{R}^{n-1} \to \mathbb{R}$, *a point* y, *a direction* d *and a line* $L(y, d) = \{y + \alpha d \mid \alpha \in \mathbb{R}\}$, *we say that* L *is of class:*

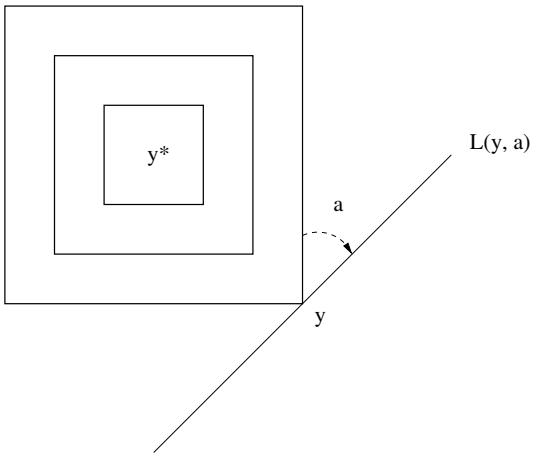

Figure 1:

L^0, *if there exist* $\delta > 0$ *such that* $F(y + \alpha d) = F(y)$ *for all* $\alpha \in I(\delta)$.

L^+, *if there exist* $\delta > 0$ *such that* $F(y + \alpha d) > F(y)$ *for some* $\alpha \in I(\delta)$.

L^-, *if there exist* $\delta > 0$ *such that* $F(y + \alpha d) \leq F(y)$ *for all* $\alpha \in I(\delta)$ *and* $F(y + \alpha d)$ *is not constant in* $I(\delta)$.

Example:
In Fig. 1, we have the level sets of a PLC function that attains its maximum at y^*. At the point y we parameterize all the possible search lines by the angle *a*, i.e., the set of possible search lines is $\{L(y, a) : a \in -[0, \pi[\}$. We encounter two types of lines: a) Lines of type L^-: Along these lines any movement from y does not improve the objective function ($a \in -[0, \pi/2]$). b) Lines of type L^+: Along these lines we can improve our objective function ($a \in -]\pi/2, \pi[$). ■

In next proposition we see that, all the lines in $\text{Proj}_y(S^k)$, the y-projection of the active linear manifold at any x^k, are of type L^0 or L^+, and thus they are good candidates for search line spaces.

Proposition 1 *If* $L(y^k, d_y)$ *is a line included in* $S^k_y = \text{Proj}_y(S^k)$, *then it is of class* L^0 *or* L^+.

Proof: By definition of active set, there exits $\delta > 0$ such that $J(x^k + \alpha d) = J(x^k)$ for all $\alpha \in I(\delta) =]-\delta, \delta[$ and for all $d \in D^k = \{d \in E^k \mid \|d\| = 1\}$. Then considering that $d = (d_y, d_z)$, $x^k = (y^k, z^k)$, we have:

$$\begin{aligned} &x^k + \alpha d \in S^k, \\ &y^k + \alpha d_y \in S^k_y, \\ &F(y^k + \alpha d_y) = z^k + \alpha d_z, \end{aligned}$$

for all $\alpha \in I(\delta)$ and for all $d \in D^k$.

If $d_z = 0$, then $F(y^k + \alpha d_y) = z^k$ for all $\alpha \in I(\delta)$, that is, $L(y^k, d_y)$ is of type L^0.

If $d_z \neq 0$, then $F(y^k + \alpha d_y) = z^k + \alpha d_z > z^k$ either for all $\alpha \in]0, \delta[$ or for all $\alpha \in]-\delta, 0[$. In the two cases $L(y^k, d_y)$ is of type L^+. ■

In view of the previous example and proposition, when using the partan algorithm, at $y^{k+\frac{1}{2}}$ we will restrict our line search along lines within $S_y^{k+\frac{1}{2}}$, the y-projection of the active linear manifold at $y^{k+\frac{1}{2}}$, since we know that they are lines of class L^0 or L^+. This is equivalent to using search directions from $E_y^{k+\frac{1}{2}} = \text{Proj}_y(E^{k+\frac{1}{2}})$. Therefore, instead of using the pure partan search direction $y^{k+\frac{1}{2}} - y^{k-1}$, we will use its projection onto $E_y^{k+\frac{1}{2}}$, i.e., at $y^{k+\frac{1}{2}}$ we will use $\text{Proj}_{E_y^{k+\frac{1}{2}}}(y^{k+\frac{1}{2}} - y^{k-1})$ as the search direction. Let us see how to compute it.

Proposition 2 *If*

$$\begin{aligned} d &= (d_y, d_z) \in \mathbb{R}^{n-1} \times \mathbb{R}, \\ A &= [A_y\, A_z] \in \mathbb{R}^{m\times(n-1)} \times \mathbb{R}^{m\times 1}, \\ E_y &= \{d_y \in \mathbb{R}^{n-1} \mid A_y d_y + A_z d_z = 0\}, \\ \widetilde{d} &\in \mathbb{R}^{n-1}, \end{aligned}$$

Then

$$\textit{Proj}_{E_y}(\widetilde{d}) = d_y^* = \left[I - A_y{}'(A_yA_y{}')^{-1}A_y\right]\widetilde{d} - \left[A_y{}'(A_yA_y{}')^{-1}A_z\right]d_z^*, \tag{9}$$

where

$$d_z^* = \frac{-1}{A_z{}'(A_yA_y{}')^{-1}A_z}\left[A_z{}'(A_yA_y{}')^{-1}A_y\right]\widetilde{d}. \tag{10}$$

Proof: By definition

$$\begin{aligned} \text{Proj}_{E_y}(\widetilde{d}) = d_y^* &= \arg\min_{d_y} \frac{1}{2}\|d_y - \widetilde{d}\|^2 \\ &\quad \text{s.t.} \quad A_y d_y + A_z d_z = 0, \end{aligned} \tag{11}$$

whose first order conditions are:

$$d_y - \widetilde{d} + A_y{}'\lambda = 0 \tag{12}$$

$$A_z{}'\lambda = 0 \tag{13}$$

$$A_y d_y + A_z d_z = 0. \tag{14}$$

The solution to this system $(d_y^*, d_z^*, \lambda^*)$ can be computed as follows. From (12) and (14) we obtain that:

$$\lambda^* = \left(A_yA_y{}'\right)^{-1}A_y\widetilde{d} + \left(A_yA_y{}'\right)^{-1}A_z d_z^*. \tag{15}$$

Pre-multiplying (15) by $A_z{}'$ and considering (13) we have that

$$d_z^* = \frac{-1}{A_z{}'(A_yA_y{}')^{-1}A_z}\left[A_z{}'(A_yA_y{}')^{-1}A_y\right]\widetilde{d}.$$

Finally, by using (15) in (12) we obtain that

$$d_y^* = \left[I - A_y{}'(A_yA_y{}')^{-1}A_y\right]\widetilde{d} - \left[A_y{}'(A_yA_y{}')^{-1}A_z\right]d_z^*.$$

■

In the following proposition we see how we can obtain an ascent feasible direction if $d_z^* \neq 0$.

Proposition 3 *a) In the previous proposition, (d_y^*, d_z^*) is an ascent direction if and only if $d_z^* > 0$.*

b) If $d_z^* \neq 0$, then for any $\gamma > 0$ we have that $d(\gamma) = (\gamma/d_z^*)(d_y^*, d_z^*)$ is an ascent feasible direction.

Proof: a) For any x we have,

$$\nabla G(x) \begin{pmatrix} d_y^* \\ d_z^* \end{pmatrix} = e_n \begin{pmatrix} d_y^* \\ d_z^* \end{pmatrix} = d_z^*.$$

b) $d(\gamma)$ is an ascent direction since $\nabla G(x) d(\gamma) = \gamma > 0$. It is feasible since by (14)

$$A\, d(\gamma) = \frac{\gamma}{d_z^*} \left[A_y \, A_z \right] \begin{pmatrix} d_y^* \\ d_z^* \end{pmatrix} = \frac{\gamma}{d_z^*} \left(A_y d_y^* + A_z d_z^* \right) = 0.$$

■

6.1 The partan face simplex method

The above discussion is summarized in the partan face simplex method, which adapts the partan strategy to the FS method in order to avoid the zig-zagging.

Partan face simplex method:

Step 0. *Step 0. Initialization:* Take $x^0 = (y^0, z^0) \in X$ such that $J(x^0) \neq \emptyset$, $\epsilon > 0$, $\gamma > 0$, $K > 0$, and set $k = 0$.

Step 1. *Compute d^k, the Rosen's gradient projection as in (4).*

Step 2. *Compute $x^{k+\frac{1}{2}} = (y^{k+\frac{1}{2}}, z^{k+\frac{1}{2}})$ by a global line search along $\text{Proj}_y(d^k)$ as in (5).*
If $k = 0$ then set $k = k + 1$ and go back to Step 1.

Step 3. *Compute the Rosen's partan projection (d_y^*, d_z^*):*

$$\begin{aligned} \widetilde{d} &= y^{k+\frac{1}{2}} - y^{k-1}, \\ A^{k+\frac{1}{2}} & \quad \text{is partitioned into } [A_y \, A_z], \\ \alpha &= A_z{}'(A_y A_y{}')^{-1} A_z. \end{aligned}$$

If $|\alpha| < \epsilon$ then set $k = 1$ and go back to Step 1,
else, compute

$$\begin{aligned} v &= A_y{}'(A_y A_y{}')^{-1} A_z, \\ d_z^* &= \frac{-1}{\alpha} v' \widetilde{d}. \end{aligned}$$

If $|d_z^*| < \epsilon$ then set $k = 1$ and go back to Step 1,
else, compute

$$d_y^* = \left[I - A_y{}'(A_y A_y{}')^{-1} A_y \right] \widetilde{d} - d_z^* v. \tag{16}$$

Step 4. *Compute $x^{k+1} = (y^{k+1}, z^{k+1})$ by a global line search along $d_y(\gamma) = (\gamma/d_z^*)d_y^*$:*

$$\begin{aligned} (y^{k+1}, z^{k+1}) = \text{arg max} \quad & z \\ \text{s.t.} \quad & (y, z) \in X \cap H^{k+\frac{1}{2}}, \end{aligned} \tag{17}$$

where

$$H^{k+\frac{1}{2}} = x^{k+\frac{1}{2}} + \text{span}\{d_y(\gamma), e_n\}.$$

Set $k = k + 1$. If $k > K$ then set $k = 0$. Go back to Step 1.

Remark 3 *a) Note that in Step 4 we use the ascent direction $d(\gamma) = (\gamma/d_z^*)(d_y^*, d_z^*)$ instead of (d_y^*, d_z^*). Of course we could use the ascent direction (d_y^*, d_z^*) whenever $d_z^* > 0$ (or its opposite if $d_z^* < 0$), but better computational results have been obtained by using $d(\gamma)$ with a constant value for γ (we do not have a clear theoretical explanation for this behavior).*

b) Also note that, after the restart procedure at the end of Step 4 (when we set $k = 0$) a pure FS iteration will be performed (for $k = 0$ the partan steps 3 and 4 are not performed). This restart procedure serves as a spacer step and therefore the convergence of the partan FS method relies on the convergence of the FS method.

7 Numerical tests

The objective of this section is to compare our prototype of the FS method to the simplex method. We test the plain version of the FS method (label 'FS') and the partan version (label 'Partan'). We study the influence of three parameters of the PLC problem: size of A, number of rows in A and shape of the PLC graph.

The FS prototype has been written in Matlab 7.0 [11]. The main parameter in the FS method are ϵ, γ and K, which have been set equal to 10^{-6}, 1 and $n-1$, respectively. The FS initial point has been set equal to 0. In our benchmark, we have used the Matlab simplex and the Mosek simplex [17]. Matlab simplex is a very basic implementation written in Matlab (interpreted language). Mosek simplex is a state-of-the-art implementation written in C (compiled language). For both simplex implementations we have used their default parameters and default starting point, except for the the stopping tolerance that has been set equal to 10^{-6}. All programs have been run on a PC (Pentium-IV, 2.8 GHz, with 4 Gb of RAM memory) under the Linux operating system.

We have performed four tests. For each test we display two tables. In the first one we report the problem description and the optima obtained. In the second one, we report the performance in terms of number of iterations and CPU time. In each performance table, for each data column we compute the average. We also compute what is labeled as 'Relative' which is a normalization of the average values and is computed as the average value divided by the Mosek average value.

7.1 Influence of the instance size

What is the impact of the instance size (size of A) for the FS method ? To answer this question, we solve ten randomly generated instances of the PLC problem (2). For each instance we

generate $3n$ random hyperplanes in $\mathbb{R}^n$ with equation:

$$\pi_j \equiv z = z_j + s'_j(y - y_j) \quad j = 1, \ldots, 3n$$

where z_j, s_j and y_j are uniformly distributed in $[-100, 100]$, $\pm[0.1, 10]^{n-1}$ and $[-100, 100]^{n-1}$, respectively. That is, the constraint matrix A associated to each generated PLC instance has dimension rows$\times$columns = $3n \times n$.

In this paper all the PLC instances have at least $m = 3n$ hyperplanes. Our objective was to restrict our computational experiments to bounded PLC functions and for this m value all the randomly generated instances resulted to be bounded (for smaller values of m, sometimes we obtained unbounded PLC instances).

As we can see in Table 2, the impact of the size of A is sever for the FS performance. This impact is greatly reduced in the partan version (by a factor greater than 10). The partan FS method clearly outperforms the Matlab simplex (over 200 times faster).This CPU time difference is remarkable since the two methods use the same numerical and computational technology: the computing of $A^{-1}b$ is done by the Matlab compiled command $A \setminus b$ and no basis factorization updating is performed in any of the two implementations. The reason for this performance speed-up is twofold: on the one hand the partan FS method needs, on average, 60% of the Matlab simplex number of iteration. On the other hand, as we can see in Table 3, the number of active hyperplanes at each partan FS iteration is relatively low, specially for the larger instances. One of the main computational effort of the FS method is to compute $(A^k A^{k'})^{-1}$, which on average has dimension 28 for the current test (partan version). At each iteration, the simplex method computes B^{-1} with dimension equal to m, which in this test is on average 330.

In sharp contrast, the Mosek simplex performs much faster than the three other solvers. Although the objective of this test is not to compare the two simplex implementation, we wish to point out three important reasons for this huge difference on the simplex CPU times: a) Matlab simplex does not perform neither LU factorization of the basis nor its updating: every linear system of the simplex method is solved from scratch! b) As we mentioned, Matlab simplex is an interpreted code and Mosek simplex is a compiled one. c) Mosek incorporates many numerical tricks (e.g. rescaling of the problem) that greatly accelerates the simplex method.

Finally, in Table 3 we observe that the average number of radar iterations is 1 or close to 1. That means that most of the time the active set at the current FS iterate is included in the active set at the next FS iterate, i.e. $J(x^k) \subset J(x^{k+1})$. This observation opens the door to linear algebra improvements as for example a cheap updating of the projection matrix P^k.

7.2 Influence of the number of constraints

It is known than the primal simplex typically requires at most $3m$ iterations (pivots) to attain optimality [22]. Can we say some thing similar about the FS method? To answer this question, we solve ten random PLC instances generated as in the previous test, with a fixed dimension (100) and the number of constraints ranging from 300 to 1200. In Tables 4-5 we report the results. Regarding the number of iterations: a) Neither FS methods nor the Mosek simplex seem to depend on the number of constraints. b) Matlab simplex number of iterations strongly depends on the number of constraints (for this test 'number of iterations' $\backsimeq 10m$). Regarding the CPU time: a) The FS methods does not seem to depend on the number of constraints. b) Matlab simplex and Mosek simplex clearly depend on the number of constraints.

Table 1: Influence of the instance size: Problem description

	Size of A		Optimum			
Instance	Columns	Rows	FS	Partan	Matlab	Mosek
plc01	20	60	-1407.645692	-1407.645692	-1407.645692	-1407.645692
plc02	40	120	-1792.810503	-1792.810503	-1792.810503	-1792.810503
plc03	60	180	-1864.664313	-1864.664313	-1864.664313	-1864.664313
plc04	80	240	-2434.437657	-2434.437657	-2434.437657	-2434.437657
plc05	100	300	-2377.648663	-2377.648663	-2377.648663	-2377.648663
plc06	120	360	-3183.970574	-3183.970574	-3183.970574	-3183.970574
plc07	140	420	-3197.949391	-3197.949391	-3197.949391	-3197.949391
plc08	160	480	-3503.452506	-3503.452506	-3503.452506	-3503.452506
plc09	180	540	-3210.295251	-3210.295251	-3210.295251	-3210.295251
plc10	200	600	-3902.402666	-3902.402666	-3902.402666	-3902.402666

Table 2: Influence of the instance size: Performance

	Iterations				CPU time (sec.)			
Instance	FS	Partan	Matlab	Mosek	FS	Partan	Matlab	Mosek
plc01	32	17	37	31	0.1	0.1	1	0.1
plc02	113	45	308	84	0.1	0.1	13	0.1
plc03	276	88	967	170	0.3	0.2	69	0.2
plc04	5754	124	1413	340	6.9	0.4	250	0.3
plc05	4392	275	2582	501	6.7	1.2	829	0.5
plc06	5391	526	3649	475	11.0	2.6	1733	0.7
plc07	5344	1997	6296	719	14.1	10.7	4488	1.3
plc08	43700	1686	9124	836	208.4	12.7	7947	1.7
plc09	90288	6392	11532	1021	709.4	44.9	15451	2.5
plc10	193000	20253	17215	1027	2850.1	166.0	25490	3.1
Average	34829	3140	5312	520	380.7	23.9	5627	1.1
Relative	67	6	10	1	346.1	21.7	5115	1

Table 3: Analysis of FS and partan line searches. 'Hyperplanes' = Averaged number of active hyperplanes per line search. 'Radar iterations' = Averaged number of radar iterations per line search.

	Hyperplanes				Radar iterations	
Instance	FS	FS	Partan	Partan	FS	Partan
plc01	10	50 %	10	50 %	1.00	1.00
plc02	26	66 %	26	66 %	1.00	1.00
plc03	18	30 %	40	67 %	1.04	1.00
plc04	39	49 %	39	49 %	1.00	1.00
plc05	30	30 %	33	33 %	1.06	1.09
plc06	27	23 %	30	25 %	1.10	1.06
plc07	37	26 %	24	17 %	1.07	1.10
plc08	23	15 %	32	20 %	1.11	1.08
plc09	22	12 %	23	13 %	1.11	1.11
plc10	24	12 %	25	13 %	1.10	1.11
Average	26	31 %	28	35 %	1.06	1.05

Table 4: Influence of the number of constraints: Problem description

	Size of A		Optimum			
Instance	Columns	Rows	FS	Partan	Matlab	Mosek
plc11	100	300	-2583.688468	-2583.688468	-2583.688468	-2583.688468
plc12	100	400	-3364.767472	-3364.767472	-3364.767472	-3364.767472
plc13	100	500	-4106.718236	-4106.718236	-4106.718236	-4106.718236
plc14	100	600	-4055.119693	-4055.119693	-4055.119692	-4055.119692
plc15	100	700	-4858.708681	-4858.708681	-4858.708681	-4858.708681
plc16	100	800	-5393.471774	-5393.471774	-5393.471774	-5393.471774
plc17	100	900	-5573.752855	-5573.752855	-5573.752855	-5573.752855
plc18	100	1000	-5550.919783	-5550.919783	-5550.919783	-5550.919783
plc19	100	1100	-5637.042493	-5637.042493	-5637.042493	-5637.042493
plc20	100	1200	-5562.779705	-5562.779705	-5562.779705	-5562.779705

Table 5: Influence of the number of constraints: Performance

	Iterations				CPU time (sec.)			
Instance	FS	Partan	Matlab	Mosek	FS	Partan	Matlab	Mosek
plc11	3580	341	2931	387	5.5	1.5	842	0.4
plc12	7037	616	3508	467	13.2	2.2	1484	0.6
plc13	2634	496	5014	474	5.4	2.3	3745	0.7
plc14	3750	811	6756	495	8.6	3.8	5914	0.8
plc15	549	191	7026	434	1.5	1.3	8406	0.8
plc16	6837	582	8394	428	19.5	3.3	14146	1.0
plc17	8576	823	9700	439	26.3	4.7	18605	1.0
plc18	9301	148	10811	451	30.9	1.5	63524	1.3
plc19	1843	556	11015	485	6.2	3.7	80226	1.5
plc20	5776	494	11635	438	21.0	3.5	114834	1.5
Average	4988	506	7679	449	13.8	2.8	31173	1.0
Relative	11.1	1.1	17.1	1	13.8	2.8	31173	1

7.3 Influence of the function shape

In the case of a (smooth) quadratic function $q(x) = -\frac{1}{2}x'Qx + b$, with Q positive definite and symmetric, it is seen that the steepest ascent method gives linear convergence with

$$E(x^{k+1}) \leq \left(\frac{\Lambda - \lambda}{\Lambda + \lambda}\right)^2 E(x^k), \tag{18}$$

where $E(x) = (1/2)(x - x^*)'Q(x - x^*)$ and, Λ and λ are the largest and smallest eigenvalues of Q, respectively [16]. That is, the more or less 'elongated' shape of the graph of q determines the speed of convergence of the steepest ascent method. In the FS method applied to the PLC problem we are maximizing a nonsmooth function by following the steepest ascent on the current face. To analyze the influence of the shape of the graph, we solve ten randomly generated PLC instances, with a fixed dimension (100) and a graph constructed as as follows. Our PLC function F is defined by taking a random collection of 300 hyperplanes from the family of tangent hyperplanes to the graph of q. The definition of q is based on a diagonal matrix Q defined by the monotone and equidistant sequence $0 < \lambda_1 < \lambda_2 < \ldots < \lambda_{100}$. This implies $\Lambda = \lambda_{100}$ and $\lambda = \lambda_1$. Arbitrarily we fix $\lambda = 1/10$ and to generate the ten different q graph shapes (ten PCL instances) we use $\Lambda = 2^1/10, 2^2/10, \ldots, 2^{10}/10$. The expression for the hyperplanes are:

$$\pi_j \equiv z = q(y_j) + \nabla q(y_j)'(y - y_j) \quad j = 1, \ldots, 300,$$

with y_j uniformly distributed in $[-100, 100]^{n-1}$.

In Tables 6-7 we report the results (in Table 6 we only use the Mosek simplex optima to validate the FS optima). In contrast with the smooth case, the graph shape does not seem to influence the performance of any of the three implementations.

In Tables 8-9 we test the FS method versus the Mosek simplex method in the potentially advantageous setting for the FS method: $\lambda = \Lambda$ and a large number of constraints. Note that, even if in the previous test the shape of the graph did not have a clear influence on the FS performance, in theory, for the case $\lambda = \Lambda$ and $m = \infty$ a single FS iteration would be enough since F will be a copy of q. For the Matlab simplex we do not report results, since only the first

Table 6: Influence of the function shape: Problem description. All instances are of size rows×columns = 300×100.

	Maximum	Optimum		
Instance	Eigenvalue	FS	Partan	Mosek
plc21	2/10	2.2956216726×10^4	2.2956216726×10^4	2.2956216726×10^4
plc22	4/10	3.7530005959×10^4	3.7530005959×10^4	3.7530005959×10^4
plc23	8/10	6.7130365536×10^4	6.7130365536×10^4	6.7130365536×10^4
plc24	16/10	1.2959614351×10^5	1.2959614351×10^5	1.2959614351×10^5
plc25	32/10	2.4991985084×10^5	2.4991985084×10^5	2.4991985084×10^5
plc26	64/10	4.9278816291×10^5	4.9278816291×10^5	4.9278816291×10^5
plc27	128/10	9.6795507723×10^5	9.6795507723×10^5	9.6795507723×10^5
plc28	256/10	1.9246016667×10^6	1.9246016667×10^6	1.9246016667×10^6
plc29	512/10	3.8775769072×10^6	3.8775769072×10^6	3.8775769073×10^6
plc30	1024/10	7.7649034444×10^6	7.7649034444×10^6	7.7649034449×10^6

Table 7: Influence of the function shape: Performance

	Iterations				CPU time (sec.)			
Instance	FS	Partan	Matlab	Mosek	FS	Partan	Matlab	Mosek
plc21	4776	1064	3829	483	7.0	3.6	360	0.5
plc22	1978	306	3511	380	2.9	1.3	319	0.5
plc23	443	126	4348	410	1.1	0.8	418	0.5
plc24	970	386	4330	523	1.7	1.8	408	0.5
plc25	1701	158	3938	395	2.7	1.3	330	0.4
plc26	1569	382	4102	444	2.7	2.6	344	0.5
plc27	2199	489	3594	441	3.6	3.6	278	0.5
plc28	2212	1045	3579	474	3.7	5.7	291	0.6
plc29	30007	784	3508	332	79.3	4.5	273	0.6
plc30	1020	1971	4707	399	1.9	9.2	401	0.4
Average	4688	671	3945	428	10.7	3.4	342	0.5
Relative	11.0	1.6	9.2	1	21.4	6.8	684	1

two instances were solved in less than 10^5 seconds. For the Mosek simplex method we give results for the PLC problem (3) and for its equivalent dual formulation (DF):

$$\min_{y \in \mathbb{R}^m} \{b'y \mid A'y = e_n, y \geq 0\}.$$

Again, we encounter an important difference between the smooth and non smooth case. For a (smooth) quadratic function, by (18), the steepest ascent method will need one single iteration whenever $\lambda = \Lambda$. In Tables 9 we observe that many FS iterations may me needed even if the (nonsmooth) PLC function has been derived from a quadratic function with $\lambda = \Lambda$).

For this particular class of PLC instances we can say that the partan FS method outperforms the simplex method since the CPU times are similar and the FS prototype is coded in Matlab. Of course, when using the simplex method, in the case of a large number of constraints a better choice is to solve the dual formulation.

Table 8: Special case: Problem description. For all the cases, the number of columns is 100 and the maximum eigenvalue 1/10

	Size of A	Optimum			
Instance	Rows	FS	Partan	Mosek	Mosek DF
plc31	1000	14022.455231	14022.455231	14022.455231	14022.455235
plc32	2000	13722.010504	13722.010504	13722.010504	13722.011171
plc33	3000	13279.100662	13279.100662	13279.100662	13279.100663
plc34	4000	13187.229932	13187.229932	13187.229932	13187.231032
plc35	5000	13012.686300	13012.686300	13012.686300	13012.686300
plc36	6000	13046.096879	13046.096879	13046.096879	13046.096879
plc37	7000	12945.660286	12945.660286	12945.660286	12945.660287
plc38	8000	12781.721013	12781.721013	12781.721013	12781.721013
plc39	9000	12793.533089	12793.533089	12793.533089	12793.533089
plc40	10000	12885.004213	12885.004213	12885.004213	12885.004213

Table 9: Special case: Performance. DF stands for 'Dual Formulation'

	Iterations				CPU time (sec.)			
Instance	FS	Partan	Mosek	Mosek DF	FS	Partan	Mosek	Mosek DF
plc31	1676	275	420	1821	5.3	1.9	1.2	0.9
plc32	4530	319	399	1788	24.1	3.3	2.6	1.4
plc33	8723	324	493	2485	63.4	4.3	4.6	2.2
plc34	2214	235	422	2803	20.8	4.0	6.0	2.7
plc35	5000	195	382	2304	55.4	4.2	7.2	2.7
plc36	3992	897	543	2518	54.1	21.9	9.4	3.0
plc37	2811	143	513	2948	43.2	3.7	11.0	3.5
plc38	3612	382	584	2547	64.8	11.4	13.3	3.5
plc39	10144	460	453	2621	199.0	15.6	13.1	3.7
plc40	591	872	492	2647	13.2	35.0	15.7	4.0
Average	4329	410	470	2448	54.3	10.6	8.4	2.8
Relative	9.2	0.9	1	5.2	6.5	1.3	1	0.3

8 Conclusions

In the framework of the Kelley cutting plane method, the face simplex (FS) method is an alternative to the simplex method to maximize a piecewise linear concave (PLC) function. It is well known that the simplex method at the current vertex takes an ascending edge and stops at the other vertex of the edge. In this paper we address the question of what would happen if instead of stopping the line search at the second vertex, one continues the line search on the whole PLC surface up to the best point? The FS method gives a plausible answer.The FS method freely explores the polyhedral surface by following the Rosen's gradient projection combined with a global line search on the whole surface.

From a theoretical point of view, we have therefore enlarged the Rosen's gradient projection steps by using the radar method to perform a global line search. The other theoretical contribution has been to adapt the partan method, a conjugate gradient technique, to the case of a PLC function (partan face simplex method). Roughly speaking, this partan version has reduced the number of the plain FS iterations by a factor of 10, by avoiding zig-zagging. The CPU time improved by a factor of 5 since each partan FS iteration amounts to two plain FS iterations in terms of computational work. Finally, although we have proved the convergence of the FS method for dimension 3, convergence for the general case remains an open question.

From a computational point of view, we have compared the FS Matlab prototypes to two representative implementations of the simplex method. Matlab simplex, a very basic interpreted implementation and Mosek simplex, a state-of-the-art compiled implementation. Compared to Matlab simplex, the performance of the partan FS prototype is very competitive, mainly due to two reasons: Matlab simplex does not use any LU basis factorization/updating. On the other hand, in the FS method the computational effort is low since only a small fraction of the constraints is considered at each iteration. Compared to the Mosek simplex method, the partan FS prototype does not seem competitive except for the case of a large number of constraints. However, we think that in the FS method there is still room for numerical improvements.

Even if the FS method does not seem to outperform the simplex method, it has two inherent advantages compared to the simplex method: First, the FS method does not need the expensive initialization of the simplex method (to find an initial vertex). Second, in the case of a PLC unconstrained function, it may happen that an optimal vertex does not exist. In contrast with the FS method, the simplex method simply will not be able to optimize such a function or will need to bound the variables in an artificial way, if it made sense to do it.

Acknowledgment

The author is thankful to Jean-Philippe Vial, Alain Haurie and Nidhi Sawhney for their comments and support at Logilab, HEC, University of Geneva.

References

[1] F. Baboneau, C. Beltran, A. B. Haurie, C. Tadonki, and J.-Ph. Vial. Proximal-accpm: a versatile oracle based optimization method. In *'Advances in Computational Economics, Finance and Management Science', to appear in 'Computational Management Science'*. Kluwer.

[2] C. Beltran and F. J. Heredia. An effective line search for the subgradient method. *Journal on Optimization Theory and Applications*, 125(1):1–18, 2005.

[3] C. Beltran-Royo. Line search for piecewise linear functions by the radar method. Technical report, Logilab, HEC, University of Geneva, 2005.

[4] Dimitri. P. Bertsekas. *Nonlinear Programming*. Ed. Athena Scientific, Belmont, Massachusetts, (USA), 2n edition, 1999.

[5] G. W. Brown and T. C. Koopmans. Computational suggestions for maximizing a linear function subject to linear inequalities. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 377–380, New York, 1951. Wiley.

[6] S. Y. Chang and K. G. Murty. The steepest descent gravitational method for linear programming. *Discrete Applied Mathematics*, 25:211–239, 1989.

[7] A. R. Conn and G. Cornuéjols. A projection method for the uncapacitated facility location problem. *Mathematical programming*, 46:373–398, 1990.

[8] Sherali H. D. and I. Al-loughani. Solving euclidean distance multifacility location problems using conjugate subgradients and line-search methods. *Computational optimization and applications*, 14:275–291, 1999.

[9] D.-Z. Du and X.-S. Zhang. Global convergence of rosen's gradient projection method. *Mathematical Programming*, 44(3):357–366, 1989.

[10] W. W. Hager and S. Park. The gradient projection method with exact line search. *Journal of Global Optimization*, 30(1):103–118, 2004.

[11] D. J. Higham and N. J. Higham. *MATLAB guide*. SIAM, Philadelphia, Pennsilvania, USA, 2000.

[12] J. B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms*, volume I and II. Springer-Verlag, Berlin, 1996.

[13] J.-B. Hiriart-Urruty and C. Lemaréchal. *Fundamentals of convex analysis*. Springer, 2000.

[14] J. E. Kelley. The cutting-plane method for solving convex programs. *Journal of the SIAM*, 8:703–712, 1960.

[15] C. Lemke. The constrained gradient method of linear programming. *SIAM*, 9(1):1–17, 1961.

[16] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, Reading, Massachusetts, USA, second edition, 1984.

[17] ApS Mosek. Version 3.1.1.42, copyright (c), 1998-2004. http://www.mosek.com.

[18] P. Neame, N. Boland, and D. Ralph. An outer approximate subdifferential method for piecewise affine optimization. *Mathematical programming*, Ser. A 87:57–86, 2000.

[19] J. Rosen. The gradient projection method for nonlinear programming, I. linear constraints. *Journal of the Society for Industrial and Applied Mathematics*, 8:181–217, 1960.

[20] B. Shah, R. Buehler, and O. Kempthorne. Some algorithms for minimizing a function of several variables. *Journal of the Society for Industrial and Applied Mathematics*, 12(1):74–92, 1964.

[21] N. Z. Shor. *Nondifferentiable optimization and polynomial problems*. Nonconvex optimization and its applications. Kluwer academic publishers, 1998.

[22] M. J. Todd. The many facets of linear programming. *Mathematical Programming*, Ser. B 91:417–436, 2002.

[23] P. Wolfe. A method of conjugate subgradients for minimizing nondifferentiable functions. *Mathematical Programming Study*, 3:145–173, 1975.

[24] X.-S. Zhang. On the convergence of rosen's gradient projection method: Three-dimensional case (in chinese). *Acta Mathematicae Applicatae Sinica*, 8, 1985.

[25] G Zoutendijk. *Methods of feasible directions*. Elsevier, Amsterdan, 1960.