

**Algorithmic construction  
of the subdifferential from  
directional derivatives**

C. Audet  
W. Hare

G-2016-12

February 2016

---

Cette version est mise à votre disposition conformément à la politique de libre accès aux publications des organismes subventionnaires canadiens et québécois.

**Avant de citer ce rapport**, veuillez visiter notre site Web (<https://www.gerad.ca/fr/papers/G-2016-12>) afin de mettre à jour vos données de référence, s'il a été publié dans une revue scientifique.

This version is available to you under the open access policy of Canadian and Quebec funding agencies.

**Before citing this report**, please visit our website (<https://www.gerad.ca/en/papers/G-2016-12>) to update your reference data, if it has been published in a scientific journal.

---

Les textes publiés dans la série des rapports de recherche *Les Cahiers du GERAD* n'engagent que la responsabilité de leurs auteurs.

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2016  
– Bibliothèque et Archives Canada, 2016

The authors are exclusively responsible for the content of their research papers published in the series *Les Cahiers du GERAD*.

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2016  
– Library and Archives Canada, 2016



# Algorithmic construction of the subdifferential from directional derivatives

**Charles Audet**<sup>a</sup>

**Warren Hare**<sup>b</sup>

<sup>a</sup> GERAD & Département de mathématiques et génie industriel, Polytechnique Montréal, Montréal (Québec) Canada, H3C 3A7

<sup>b</sup> Mathematics, Irving K. Barber School, University of British Columbia, Okanagan Campus, Kelowna (B.C.) Canada, V1V 1V7

charles.audet@gerad.ca  
warren.hare@ubc.ca

**February 2016**

**Les Cahiers du GERAD  
G–2016–12**

Copyright © 2016 GERAD

**Abstract:** The subdifferential of a function is a generalization for nonsmooth functions of the concept of gradient. It is frequently used in variational analysis, particularly in the context of nonsmooth optimization. The present work proposes algorithms to reconstruct a polyhedral subdifferential of a function from the computation of finitely many directional derivatives. We provide upper bounds on the required number of directional derivatives when the space is  $\mathbb{R}^1$  and  $\mathbb{R}^2$ , as well as in  $\mathbb{R}^n$  where subdifferential is known to possess at most three vertices.

**Key Words:** Subdifferential, directional derivative, polyhedral construction, Geometric Probing.

---

**Acknowledgments:** Work of the first author was supported by Natural Sciences and Engineering Research Council of Canada (NSERC), Discovery grant #2015-05311. Work of the second author was supported by NSERC, Discovery Grant #355571-2013.

## 1 Introduction

The *subdifferential* of a nonsmooth function represents the set of generalized gradients for the function (a formal definition appears in Section 2). It can be used to detect descent directions [16, Thm 8.30], check first order optimality conditions [16, Thm 10.1], and create cutting planes for convex functions [16, Prop 8.12]. It shows up in numerous algorithms for nonsmooth optimization: steepest descent [12, Sec XII.3], projective subgradient methods [12, Sec XII.4], bundle methods [12, Sec XIV.3], etc. Its calculus properties have been well researched [16, Chpt 10], and many favourable rules have been developed. Overall, it is reasonable to state that the subdifferential is one of the most fundamental objects in variational analysis.

Given its role in nonsmooth optimization, it is no surprise that some researchers have turned their attention to the question of how to approximate a subdifferential using ‘simpler information’. Besides the mathematical appeal of such a question, such research has strong links to the fields of *Derivative-free Optimization* and *Geometric Probing*.

Derivative-free optimization focuses on the development of algorithms to minimize a function using only function values. Thus, in this case, ‘simpler information’ takes the form of function values. The ability to use function evaluations to approximate generalized gradient information is at the heart of the convergence analyses of many derivative-free optimization methods [1, 2, 8, 9]. Some research has explicitly proposed methods to approximate subdifferential for the purposes of derivative-free optimization methods [3, 4, 14, 11, 10]. Many of these researchers focus on approximating the subdifferential by using a collection of approximate *directional derivatives* [3, 4, 14]. Recall that directional derivative provides the slope for a function in a given direction in the classical limiting sense of single variable calculus – a formal definition appears in Section 2. Directional derivatives are intimately linked to subdifferential maps (see Section 2) and are appealing in that they can be approximated by simple finite difference formulae. This makes them obvious as a tool to approximate subdifferentials. However there is a drawback to this approach, as it is unclear how many directional derivatives are needed to create an accurate subdifferential approximation.

Geometric Probing considers problems of determining the geometric structure of a set by using a probe [19, 17]. If the set is a subdifferential and the probe is a directional derivative, then the Geometric Probing problem is to reconstruct the subdifferential using the ‘simpler information’ of directional derivatives (details appear in Subsection 2.1). Geometric Probing first arose in the area of robotics, where tactile sensing is used to determine the shape of an object [6]. The problem has since been well studied in  $\mathbb{R}^2$  [6, 15] and partly studied in  $\mathbb{R}^3$  [13]. As Geometric Probing principally arises in robotics and computer vision, it is not surprising that literature outside of  $\mathbb{R}^2$  and  $\mathbb{R}^3$  appears absent.

In this paper we examine the links between directional derivatives and the ability to use them to reconstruct a subdifferential. We focus on the easiest case, where the subdifferential is a polytope and the directional derivatives are exact. We show that,

- in  $\mathbb{R}^1$ , the subdifferential can be reconstructed using 2 directional derivative evaluations (Section 3);
- in  $\mathbb{R}^2$ , if the subdifferential is a polytope known to have  $n_v$  vertices, then the subdifferential can be reconstructed using  $3n_v$  directional derivative evaluations (Section 4);
- in  $\mathbb{R}^2$ , if the subdifferential is a polytope with  $n_v$  vertices, but  $n_v$  is unknown, then the subdifferential can be reconstructed using  $3n_v + 1$  directional derivative evaluations (Section 4);
- in  $\mathbb{R}^n$ , if the subdifferential is a polytope known to have at most 2 vertices, then the subdifferential can be reconstructed using  $3n - 1$  directional derivative evaluations (Subsection 5.2); and
- in  $\mathbb{R}^n$ , if the subdifferential is a polytope known to have at most 3 vertices, then the subdifferential can be reconstructed using  $5n - 1$  directional derivative evaluations (Subsection 5.3).

These results can be loosely viewed as providing a lower bound on the number of approximate directional derivative evaluations that would be required to create a good approximation of the subdifferential in derivative-free optimization. The results also advance research in Geometric Probing, which historically has only considered polytopes in  $\mathbb{R}^2$  and  $\mathbb{R}^3$ .

Before proving these results, Section 2 proposes the necessary background for this work. It also includes a method of problem abstraction which links the research to Geometric Probing. Sections 3 and 4 present the results in  $\mathbb{R}^1$  and in  $\mathbb{R}^2$ , and Section 5 focuses on  $\mathbb{R}^n$ . The paper concludes with some thoughts on the challenge of reconstructing polyhedral subdifferentials when directional derivatives are only available via finite difference approximations, and some other possible directions for future research.

## 2 Definitions and problem abstraction

Given a nonsmooth function  $f : \mathbb{R}^n \mapsto \mathbb{R} \cup \{+\infty\}$  and a point  $\bar{x} \in \mathbb{R}^n$  where  $f(\bar{x})$  is finite, we define the *regular subdifferential* of  $f$  at  $\bar{x}$ ,  $\partial f(\bar{x})$ , as the set

$$\partial f(\bar{x}) = \{v \in \mathbb{R}^n : f(x) \geq f(\bar{x}) + v^\top(x - \bar{x}) + o|x - \bar{x}|\}.$$

If  $f$  is convex, then this is equivalent to the classical subdifferential of convex analysis [16, Prop 8.12], i.e.,

$$\text{if } f \text{ is convex, then } \partial f(\bar{x}) = \{v \in \mathbb{R}^n : f(x) \geq f(\bar{x}) + v^\top(x - \bar{x}) \text{ for all } x \in \mathbb{R}^n\}.$$

In this paper, we consider the situation where the subdifferential is a polyhedral set. This arises, for example, when  $f$  is a *finite max function*. In particular, if

$$f = \max \{f_i : i \in \{1, 2, \dots, n_f\}\}, \quad (1)$$

where each  $f_i \in \mathcal{C}^1$ , then

$$\partial f(\bar{x}) = \text{conv}\{\nabla f_i(\bar{x}) : i \in A(\bar{x})\} \quad (2)$$

where  $A(\bar{x}) = \{i \in \{1, 2, \dots, n_f\} : f_i(\bar{x}) = f(\bar{x})\}$  [16, Ex 8.31]. It follows that, in this case, the subdifferential  $\partial f(\bar{x})$  is a (nonempty) polytope with at most  $n_f$  vertices.

Related to the subdifferential is the *directional derivative*. Formally, the directional derivative  $df(\bar{x}; \bar{d})$  of a continuous function  $f : \mathbb{R}^n \mapsto \mathbb{R}$  at  $\bar{x} \in \mathbb{R}^n$  in the direction  $\bar{d} \in \mathbb{R}^n$  is

$$df(\bar{x}; \bar{d}) = \lim_{\tau \searrow 0} \frac{f(\bar{x} + \tau \bar{d}) - f(\bar{x})}{\tau}.$$

Given a nonsmooth function  $f : \mathbb{R}^n \mapsto \mathbb{R} \cup \{+\infty\}$  and a point  $\bar{x}$  where  $f(\bar{x})$  is finite,  $df(\bar{x}; \bar{d})$  is defined via

$$df(\bar{x}; \bar{d}) = \liminf_{\tau \searrow 0, d \rightarrow \bar{d}} \frac{f(\bar{x} + \tau d) - f(\bar{x})}{\tau}.$$

Directional derivatives are linked to the subdifferential through the following classical formula

$$\partial f(\bar{x}) = \{v : v^\top d \leq df(\bar{x}; d) \text{ for all } d\} \quad [16, \text{Ex 8.4}]. \quad (3)$$

Thus, given all possible directional derivatives, one can recreate the subdifferential as the infinite intersection of halfspaces,

$$\partial f(\bar{x}) = \bigcap_{d: \|d\|=1} \{v \in \mathbb{R}^n : v^\top d \leq df(\bar{x}; d)\}.$$

Using this to develop a numerical approach to construct the subdifferential is, in general, impractical. However, if the subdifferential is a polytope, then it may be possible to reconstruct the exact subdifferential using a finite number of directional derivative evaluations.

In general we will consider two basic cases:

- I-  $\partial f(\bar{x})$  is a polytope, and an upper bound  $n_f$  on its number of vertices  $n_v$  is known;
- II-  $\partial f(\bar{x})$  is a polytope, but no upper bound on the number of vertices is available.

Case I corresponds to the situation where  $f$  is a finite max function and the number of sub-functions  $n_f$  used in constructing  $f$  is known. Case II corresponds to the case where  $f$  is a finite max function, but no information about the function is available. We shall see that the algorithms for both cases is the same, but Case I provides the potential for early termination.

## 2.1 Links to Geometric Probing

Under our assumption that  $\partial f(\bar{x})$  is a nonempty polytope, and in light of Equation (3), we reformulate the problem in the following abstract manner:

*Working in  $\mathbb{R}^n$ , the goal is to find all vertices  $X_v = \{v^1, v^2, \dots, v^{n_v}\}$  of a nonempty compact polytope  $X$  using an oracle  $D : \mathbb{R}^n \rightarrow \mathbb{R}$  that returns the value  $D(d) = \max_{v \in X} v^\top d$ .*

In general, polytopes will be denoted by a capital letter variable ( $X, P, S$ ) and their corresponding sets of vertices will be denoted using a subscript  $v$  (e.g.,  $X_v, P_v, S_v$ ). In  $\mathbb{R}^2$ , we shall denote the edges of polytope  $X$  by using a subscript  $e$ :  $X_e = X \setminus (\text{int}(X) \cup X_v)$ . Linking to constructing subdifferentials is achieved by setting  $X = \partial f(\bar{x})$  and  $D(d) = df(\bar{x}; d)$ . The assumptions for our two cases can be written as

- I-  $X$  is a nonempty polytope with  $n_v$  vertices, and  $n_f \geq n_v$  is given,
- II-  $X$  is a nonempty polytope with  $n_v$  vertices, and  $n_f = \infty$  is given.

The interest in this reformulation is that it almost exactly corresponds to what is known as the *Hyperplane Probing* problem in the field of Geometric Probing. In Hyperplane Probing the goal is to determine the shape of a polyhedral set in  $\mathbb{R}^2$  given a hyperplane probe which measures the first time and place that a hyperplane moving parallel to itself intersects the polyhedral set [7, 19]. In Hyperplane Probing, it is generally assumed that 0 is in the interior of the set, although this is only for convenience and has no effect on algorithm design [7].

Hyperplane Probing dates back to at least 1986 [7], where it was shown to be the dual problem to *Finger Probing* (where the probe measures the point where a ray exits the polyhedral set. Using this knowledge, it was proven that to fully determine a polyhedral set with  $n_v$  vertices,  $3n_v - 1$  probes are necessary, and  $3n_v$  probes are sufficient [7]. Some variants of Hyperplane Probing exist. In 1986, Bernstein considered the case when the polyhedral set  $X$  is one of a finite list of potential sets  $X \in \{X^1, X^2, \dots, X^p\}$  [5]. In this case the number of probes can be reduced to  $2n_v + p$ , where  $n_v$  is the number of vertices of  $X$  and  $p$  is the size of the potential list of polyhedral sets. In another variant, a double hyperplane probe is considered, which provides both the first and last place that a hyperplane moving parallel to itself intersects the polyhedral set [18]. This extra information allows the resulting algorithm to terminate after  $3n_v - 2$  probes.

Our problem differs from Hyperplane Probing in two small ways. First, instead of providing the first time *and place* that a hyperplane moving parallel to itself intersects the polyhedral set, our assumptions provide an oracle that yields only the first time. Interestingly, this reduction of information has very little impact on the algorithm or convergence. Indeed, we find it is sufficient to use  $3n_v + 1$  oracle calls (as apposed to  $3n_v$  for Hyperplane Probing). In our case, the extra call is required to confirm that the final suspected vertex is indeed a vertex. Second, we consider the space of polytopes  $\mathbb{R}^n$ , instead of polyhedral sets in  $\mathbb{R}^2$ . While some recent research has examined Hyperplane Probing in  $\mathbb{R}^3$  [13], to our knowledge no research has explored the most general case of  $\mathbb{R}^n$ . It is worth noting that the original work of Dobkin, Edelsbrunner, and Yap [7] defined Hyperplane Probing as a problem in  $\mathbb{R}^n$ , but only studied the problem in  $\mathbb{R}^2$ .

## 2.2 Notation

Using the oracle notation of Geometric Probing, we introduce the following notation. The vector  $e_j \in \mathbb{R}^n$  denotes the unit vector in the direction of the  $j^{\text{th}}$  coordinate. For a vector  $d \in \mathbb{R}^n$  and a value  $D(d)$ , we define the *generated constraint halfspace* by

$$H(d) = \{v \in \mathbb{R}^n : v^\top d \leq D(d)\}$$

and the *generated hyperplane* by

$$L(d) = \{v \in \mathbb{R}^n : v^\top d = D(d)\}.$$

Finally, given a set of vectors  $D = \{d^1, d^2, \dots, d^m\}$  and corresponding values, we define the *generate constraint set* by

$$\mathcal{H}(D) = \bigcap_{d \in D} H(d).$$

### 3 One-dimensional space

When working in  $\mathbb{R}^1$  the problem is trivially solved. Indeed, in  $\mathbb{R}^1$ , the number of vertices of  $X$  must be either 1 or 2, i.e., the polytope  $X$  will either be a single point, or a close interval.

If  $n_v$  is known to be equal to 1, then a single evaluation suffices:  $X = \{D(1)\}$ . If  $n_v$  is unknown or is known to be 2, then exactly two evaluations suffice. Specifically, evaluate  $D(1)$  and  $D(-1)$ ; if both are equal then  $n_v = 1$  and  $X = \{D(1)\}$ , otherwise  $n_v = 2$  and  $X = \text{conv}\{D(1), D(-1)\}$ .

### 4 Two-dimensional space

In  $\mathbb{R}^2$ , the problem becomes more difficult, as  $n_v$  or its upper bound  $n_f$  can take on any positive integer value. Our purposed algorithm (Algorithm 4.1 below) continually refines two polyhedral approximations of  $X$ . The set  $P$  is a polyhedral outer approximation of  $X$ , and the set  $S$  is a polyhedral inner approximation of  $X$ . The outer approximation set is initialized with a triangle, and the inner approximation is initialized as the empty set. The algorithm proceeds by carefully truncating vertices of  $P$  until a vertex of  $P$  is proven to be a true vertex of  $X$ . As vertices of  $P$  are found to be true vertices of  $X$ , they are added to  $S$ . The algorithm terminates when  $P = S$  or when the cardinality of  $S_v$  equals  $n_f$ .

In the algorithm below, recall we denote the edge set of  $P$  by  $P_e$  and the vertex set of  $P$  by  $P_v$ . Similarly,  $S_e$  is the edge set of  $S$  and  $S_v$  is the vertex set of  $S$ .

**Algorithm 4.1** Finding  $X_v$  in  $\mathbb{R}^2$ ,

given an oracle  $D(d) = \max_{v \in X} v^\top d$ , and

given an upper bound on the number of vertices:  $n_f \in \mathbb{N} \cup \{\infty\}$ ,  $n_f \geq n_v$ .

Initialize:

Define the initial outer approximation polytope

$$P = \mathcal{H}(D) \text{ with } D = \{e_1, e_2, -e_1 - e_2\}.$$

If  $P$  is a singleton, then set  $S_v = P$  and terminate.

Otherwise, determine the 3 vertices of  $P$  and enumerate them clockwise

$$P_v = \{p^1, p^2, p^3\}.$$

Create the (empty) initial inner approximation, and initialize counter

$$S_v = \emptyset \text{ and set } i = 1.$$

While  $P_v \neq S_v$  and  $|S_v| < n_f$

Set  $a = p^i$ ,  $b = p^{i+1}$ , and  $c = p^{i+2}$ , working modularly if  $i + 2 > |P_v|$ .

Choose  $d \in \mathbb{R}^2$  such that  $d^\top a = d^\top c < d^\top b$ .

Compute  $D(d) \in [d^\top a, d^\top b]$ .

• If  $D(d) = d^\top b$ , then  $b \in X_v$ :

update  $S_v \leftarrow S_v \cup \{b\}$ ,  $P_v \leftarrow P_v$ ,

$i \leftarrow i + 1$ .

• If  $D(d) = d^\top a$ , then  $\{a, c\} \in X_v$ :

update  $S_v \leftarrow S_v \cup \{a, c\}$ ,  $P_v \leftarrow P_v \setminus \{b\}$ ,

re-enumerate  $P_v$  starting at  $a$  working clockwise,

$i \leftarrow i + 1$ .

• Otherwise (if  $d^\top a = d^\top c < D(d) < d^\top b$ ), then two new potential vertices are discovered:

compute the two distinct intersection points  $\{b', c'\} = P_e \cap L(d)$ ,

update  $S_v \leftarrow S_v$ ,  $P_v \leftarrow (P_v \setminus \{b\}) \cup \{b', c'\}$ ,

re-enumerate  $P_v$  starting at  $a$  working clockwise,

$i \leftarrow i$ .

Return  $S_v$ .

Before examining the algorithm's convergence properties, we provide an illustrative example.

**Example 4.2** Consider the polyhedral set  $X$  in Figure 1, a triangle in  $\mathbb{R}^2$ .

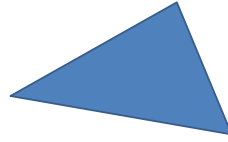


Figure 1: A polyhedral set  $X$  in  $\mathbb{R}^2$  used for Example 4.2.

Using the notation of Subsection 2.2,  $P$  is initialized as  $P = \mathcal{H}(D)$  with  $D = \{e_1, e_2, -e_1 - e_2\}$ . This creates a triangle containing  $X$ . If the triangle was degenerate, i.e.,  $P$  was a singleton, then the problem would be solved:  $X = P$ . In this example,  $P$  is not degenerate, hence  $P$  has exactly 3 extreme points, which we label  $\{p^1, p^2, p^3\}$ , see Figure 2. Notice that  $P$  would still be a triangle if the set  $X$  were a line segment rather than a triangle.

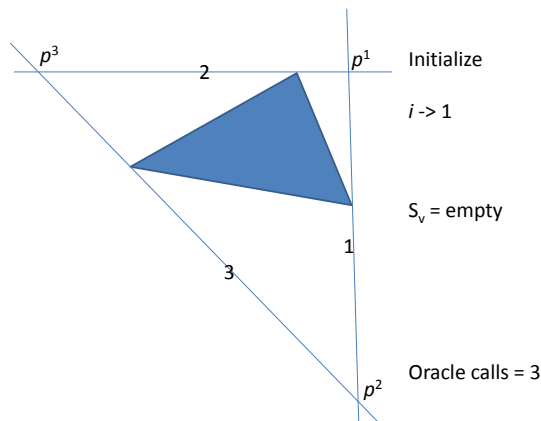


Figure 2: Initialization of Algorithm 4.1. Numbers on the lines represent which oracle call created the tangent plane.

Setting  $i = 1$ , the algorithm creates a hyperplane parallel to the line segment adjoining  $p^1$  and  $p^3$  of Figure 2:  $d$  is set to  $-e_2$ . The fourth oracle call is used to check for new potential vertices. In this case  $d^\top a < D(d) < d^\top b$ , so two new potential vertices are discovered. As a result, the index  $i$  is unchanged and only  $P_v$  is updated. The result appears in Figure 3 in which the indices of the vertices are relabelled.

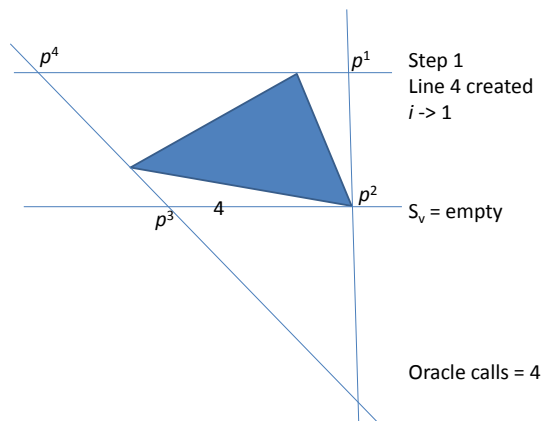


Figure 3: Step 1 of Algorithm 4.1. The new tangent plane is labelled with a 4 (the fourth oracle call).

The reordered vertices in  $P_v$  are now used to create a new hyperplane parallel to the line segment adjoining  $p^1$  and  $p^3$  (note  $p^3$  has changed in Figure 3, so this line is distinct from the line in step 1). In this case the result is  $D(d) = d^\top b$ , where  $b = p^2$ . As a result, the index  $i$  is incremented to 2, and  $p^2$  is the first vertex to be discovered. It is placed into  $S_v = \{p^2\}$ . The result appears in Figure 4.

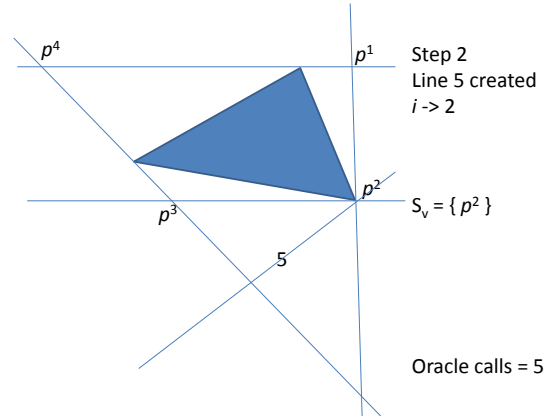


Figure 4: Step 2 of Algorithm 4.1. The new tangent plane is labelled with a 5 (the fifth oracle call).

In step 3, the new hyperplane is parallel to the segment joining  $p^2$  and  $p^4$  of Figure 4. Like step 1, the result is two new potential vertices, and  $i$  remains unchanged, see Figure 5.

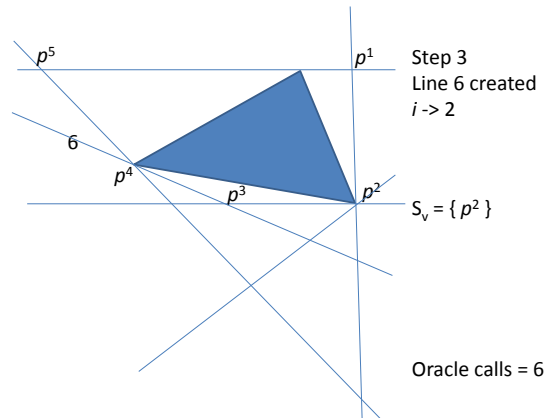


Figure 5: Step 3 of Algorithm 4.1. The new tangent plane is labelled with a 6 (the sixth oracle call).

Step 4, Figure 6, discovers vertex  $p^3$  with an hyperplane tangent to one of the sides of the triangle. The index  $i$  is incremented, and  $p^3$  is added to  $S_v$ .

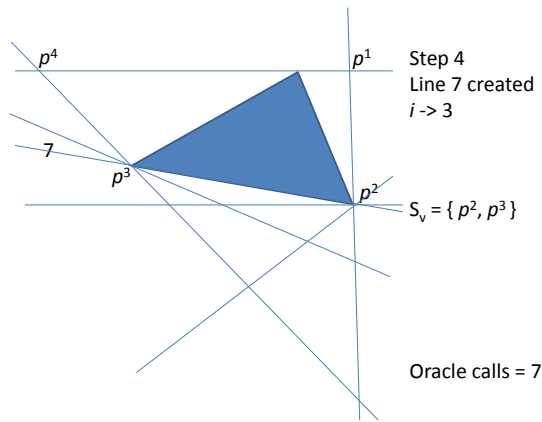


Figure 6: Step 4 of Algorithm 4.1. The new tangent plane is labelled with a 7 (the seventh oracle call).

Step 5, Figure 7, creates two new potential vertices with an hyperplane parallel to the line segment joining the  $p^3$  and  $p^1$  from step 4. This is an example of using modularity.

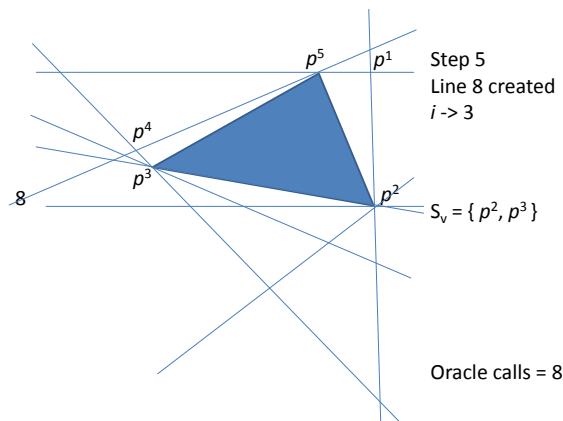


Figure 7: Step 5 of Algorithm 4.1. The new tangent plane is labelled with a 8 (the eighth oracle call).

In Step 6, Figure 8, we have  $d^\top a = d^\top c = D(d)$ , so two vertices are discovered simultaneously. (In this case, vertex  $p^3$  was actually already known, but this is not necessarily always the case.) This results in  $p^5$  being relabelled as  $p^4$  and being added to  $S_v$ .

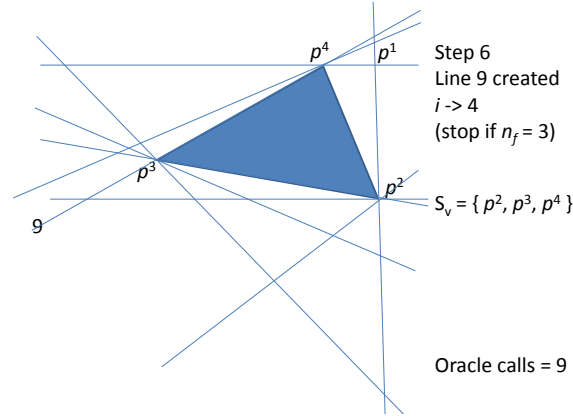


Figure 8: Step 6 of Algorithm 4.1. The new tangent plane is labelled with a 9 (the ninth oracle call).

If  $n_f = 3$ , then the algorithm can stop at this point, as  $S_v$  contains three known vertices. Conversely, if  $n_f > 3$ , then the algorithm requires one more step to truncate  $p^1$  from the potential vertex list, see Figure 9.

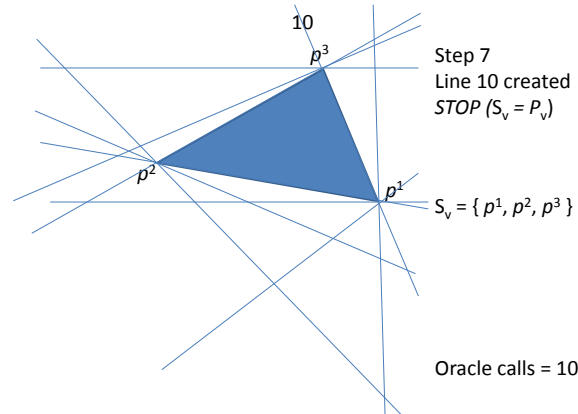


Figure 9: Step 7 of Algorithm 4.1. The new tangent plane is labelled with a 10 (the tenth oracle call).

In summary, the algorithm uses either a total of  $3n_v = 9$  oracle calls if  $n_f = 3$ , or  $3n_v + 1 = 10$  oracle calls if  $n_f > 3$ . In both situations it returns  $S_v = X_v$ .

Example 4.2 demonstrates the ideas behind the algorithm, and shows that it is possible to require  $3n_v + 1$  oracle calls. The next theorem proves the algorithm converges to the correct vertex set. It also proves that, if  $n_f = n_v$ , then at most  $3n_v$  oracle calls will be required, and if  $n_f > n_v$ , then at most  $3n_v + 1$  oracle calls will be required.

**Theorem 4.3 (Convergence of Algorithm 4.1)** Let  $X$  be a polytope in  $\mathbb{R}^2$  with  $n_v$  vertices contained in set  $X_v$ .

If  $n_v = 1$ , then Algorithm 4.1 terminates after the initialization step.

If  $n_f = n_v$ , then Algorithm 4.1 terminates after at most  $3n_v$  oracle calls.

If  $n_f > n_v$ , then Algorithm 4.1 terminates after at most  $3n_v + 1$  oracle calls.

In either case, when the algorithm terminates with  $S_v = X_v$ .

**Proof.** We shall use the notation of Subsection 2.2. First note, if  $X_v$  is a singleton, then the initialization step will result in  $P = X_v$  and the algorithm terminates after 3 oracle calls.

If  $X_v$  is not a singleton, then each oracle call of the algorithm,  $D(d)$  introduces a new tangent plane to  $X$ . Specifically

$$L(d) = \{v \in \mathbb{R}^2 : v^\top d = D(d)\}$$

is a tangent plane to  $X$ . As  $X$  is polyhedral, we must have  $L(d) \cap X_v \neq \emptyset$ . Let  $v \in L(d) \cap X_v$ . The vertex  $v \in X_v$  lies in one of three sets: the interior, the edges or the vertices of  $P$ .

If  $v \in \text{int}P$ , then the previously undiscovered vertex  $v$  of  $X_v$  has been added to  $P_e$ . As  $X$  has  $n_v$  vertices, this can happen at most  $n_v$  times.

If  $v \in P_e$ , then it will be shifted from  $P_e$  into the potential vertex set  $P_v$ . Again, as  $X$  has  $n_v$  vertices, this can happen at most  $n_v$  times.

Finally, if  $v \in P_v$ , then we are in the case of  $D(d) = d^\top b$  or  $D(d) = d^\top a$ , so the potential vertex  $v \in P_v$  has been confirmed as a true vertex of  $X$  and placed in  $S_v$ . This can happen at most  $n_v$  times.

Thus, after at most  $3n_v$  oracle calls,  $S_v$  will contain all  $n_v$  vertices of  $X$ . If  $n_f = n_v$ , then the algorithm will terminate at this point.

If  $n_f > n_v$  and  $p^1 \notin X_v$ , then after at most  $3n_v$  oracle calls,  $S_v = X_v$ , but  $P_v$  may still contain  $p^1$ . One final oracle call will remove  $p^1$  from  $P_v$ , making  $S_v = P_v$  and the algorithm will terminate.  $\square$

In some situations it is possible to terminate the algorithm early.

**Lemma 4.4 (Improved stopping when  $n_f = n_v$ )** *Let  $X$  be a polytope with  $n_v$  vertices contained in set  $X_v$ . Suppose  $n_f = n_v$ . Suppose the algorithm has run to the point where  $n_v - 1$  vertices are identified. If  $P_e$  contains two edges that are not adjacent to any of the known vertices, then the intersection of those two edges must be the final vertex. As such, the algorithm can be terminated.*

Lemma 4.4 is particularly useful when  $n_f = n_v = 2$ .

**Corollary 4.5 (Special case of  $n_f = n_v = 2$ )** *Let  $X$  be a polytope with  $n_v = 2$  vertices contained in set  $X_v$ . If  $n_f = 2$ , then the algorithm can be terminated after just 5 oracle calls.*

**Proof.** Following the logic in the proof of Theorem 4.3, at most 2 oracle calls can move a vertex from the interior of  $P$  to the edge set of  $P$ , and at most 2 oracle calls can move a vertex from the edge set of  $P$  to the vertex set of  $P$ . Therefore, after 5 oracle calls, at least one vertex has been identified. If two vertices are identified, then we are done. Otherwise, we must be in the situation shown in Figure 10.

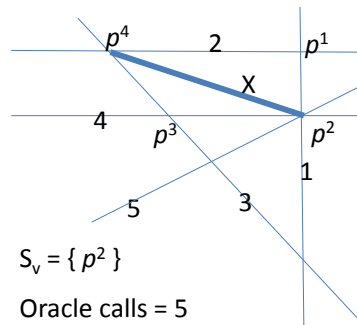


Figure 10: The situation after 5 oracle calls if  $n_v = n_f = 2$  and only one vertex has been identified. By Lemma 4.4, potential vertex  $p^4$  must be a true vertex.

In particular, we must have one vertex of  $P$  that has 3 lines through it, one of which is redundant in defining  $P$ . The other 2 lines that make up  $P_e$  must not intersect this vertex, and cannot create a vertex of  $P$  with 3 lines through it. The only possible way to do this is a quadrilateral. Lemma 4.4 now applies, so the final vertex can be identified without an additional oracle calls.  $\square$

It is worth noting that, unless the special termination trick in Lemma 4.4 applies, then the bounds provided in Theorem 4.3 are tight, as was demonstrated in Example 4.2.

It is clear that there is nothing particularly special about the directions  $e_1, e_2$ , and  $-e_1 - e_2$  used in the initialization of Algorithm 4.1. If these directions are replaced by any three directions that *positively span*  $\mathbb{R}^2$ , then the algorithm behaviour is essentially unchanged. More interestingly, the initialization set can be replaced by any set that positively spans  $\mathbb{R}^2$ , with the only negative impact being the potential to waste oracle calls during the initialization phase. The following lemma analyses this situation and will be referred to later in the paper.

**Lemma 4.6** *Suppose the initialization set  $\{e_1, e_2, -e_1 - e_2\}$  in Algorithm 4.1 is replaced by  $D = \{d^1, d^2, \dots, d^m\}$ , where  $D$  positively spans  $\mathbb{R}^2$  in the sense that*

$$\{x : x = \sum_{i=1}^m \lambda_i d^i, \lambda^i \geq 0\} = \mathbb{R}^2.$$

*If  $n_f = n_v$ , then Algorithm 4.1 terminates after at most  $(3n_v) + (m - 3)$  oracle calls.*

*If  $n_f > n_v$ , then Algorithm 4.1 terminates after at most  $(3n_v + 1) + (m - 3)$  oracle calls.*

*In either case, when the algorithm terminates we have  $S_v = X_v$ .*

**Proof.** Since  $\{x : x = \sum_{i=1}^m \lambda_i d^i, \lambda^i \geq 0\} = \mathbb{R}^2$ , the initialization directions allow the algorithm to create a compact initialization polytope  $P$  that contains  $X$ . If a vertex of  $P$  is defined by 4 or more oracle, then any oracle call past the first 3 is potentially wasted. However, all other oracle calls follow the same rules as those in the proof of Theorem 4.3. The bounds follow from the fact that a maximum of  $m - 3$  oracle calls will be wasted.  $\square$

We conclude this section with a remark that will be important in analyzing Algorithm 5.3.

**Remark 4.7** *If the initialization set  $D = \{e_1, e_2, -e_1, -e_2\}$  is used and  $n_v \geq 2$ , then no oracle calls will be wasted in creating the initialization set. Thus, in this case, the exact bounds of Theorem 4.3 will still hold. (If  $n_v = 1$ , then the resulting will use 4 oracle calls instead of 3, but the algorithm will still terminate immediately after the initialization step.)*

## 5 $n$ -dimensional space with at most 3 vertices

This section is devoted to  $\mathbb{R}^n$  in which the polytope  $X$  has at most  $n_f \leq 3$  vertices. We consider 3 subcases.

### 5.1 $n$ -dimensional subcase with $n_f = 1$

The simplest case in  $\mathbb{R}^n$  occurs when the upper bound on the number of vertices of  $X$  is  $n_f = 1$ . This trivial case is solved by  $n$  calls to the oracle  $D$ : For  $i = 1, 2, \dots, n$ , evaluate  $D(e_i)$  and return  $X_v = \{D(e_1)e_1 + D(e_2)e_2 + \dots + D(e_n)e_n\}$ .

### 5.2 $n$ -dimensional subcase with $n_f = 2$

The next simplest case is when the upper bound is  $n_f = 2$ . We present an alternate algorithm for this case, which uses a similar approach to constructs an outer approximation  $P$  of  $X$ . However, in this case a hyperrectangle is used in the initialization phase to bound the vertices.

**Algorithm 5.1** Finding  $X_v$  in  $\mathbb{R}^n$ ,  
 given an oracle  $D(d) = \max_{v \in X} v^\top d$ , and  
 given the upper bound on the number of vertices is  $n_f = 2 \geq n_v$ .

Initialize:

Define the initial outer approximation polytope hyperrectangle  
 $P = \mathcal{H}(D)$  with  $D = \{\pm e_i : i = 1, 2, \dots, n\}$ .  
 If  $P$  is a singleton, then return  $P$  and terminate.  
 Otherwise, set  $\ell_i = -D(-e_i)$  and  $u_i = D(e_i)$  for all  $i$ .  
 By relabelling indices if necessary, assume  $\ell_1 < u_1$ .  
 Initialize points  $a = (\ell_1, \ell_2, \dots, \ell_n)$ ,  $b = (u_1, u_2, \dots, u_n)$ .

For  $i$  from 2 to  $n$

If  $\ell_i < u_i$ , then  
 choose  $d \in \mathbb{R}^n$  with  $d_i = 1$  and  $d_j = 0$  for all indices  $j > i$   
 such that  $d^\top a = d^\top b$ ;  
 if  $D(d) > d^\top a$ , then reset  $a_i \leftarrow u_i$  and  $b_i \leftarrow \ell_i$ .

Return  $\{a, b\}$ .

**Theorem 5.2 (Convergence of Algorithm 5.1)** *Let  $X$  be a polytope with  $n_v \in \{1, 2\}$  vertices. Let the known upper bound for  $n_v$  be  $n_f = 2$ .*

*If  $n_v = 1$ , then Algorithm 5.1 terminates after exactly  $2n$  oracle calls with the singleton  $X_v = P$ .*

*If  $n_v = 2$ , then Algorithm 5.1 terminates after at most  $3n - 1$  oracle calls with  $X_v = \{a, b\}$ .*

**Proof.** The initialization phase of the algorithm calls the oracle exactly  $2n$  times, providing bounds  $\ell_i \leq x_i \leq u_i$  for each index  $i$ . It is obvious that  $X$  is a singleton if and only if  $\ell_i = u_i$  for all  $i \in \{1, 2, \dots, n\}$ , in which case  $X = P$ . Otherwise the algorithm proceeds into the iterative loop, that identifies the components of the two vertices of  $X_v$ .

For each  $i \in \{2, \dots, n\}$ , the bound  $\ell_i$  is initially assigned to  $a_i$ , and  $u_i$  is initially assigned to  $b_i$ . If both lower and upper bounds are equal,  $\ell_i = u_i$ , then both  $a_i$  and  $b_i$  remain at this value. When they differ, an additional call to the oracle will indicate to which of  $a_i$  or  $b_i$  will the bounds be associated. The algorithm constructs the vector  $d \in \mathbb{R}^n$  in such a way that  $d^\top a = d^\top b$ , which implies that  $\sum_{k=1}^{i-1} d_k a_k + \ell_i = \sum_{k=1}^{i-1} d_k b_k + u_i$ . Therefore, if  $D(d) = d^\top a$ , then it follows that the lower bound was correctly attributed to  $a_i$  and the upper bound was correctly attributed to  $b_i$ . Otherwise, they need to be swapped:  $a_i \leftarrow u_i$  and  $b_i \leftarrow \ell_i$ .

The total number of calls to the oracle is  $2n$  for initialization step plus a maximum of  $n - 1$  from the loop of  $i = 2$  to  $n$ . Thus, an overall maximum of  $3n - 1$  oracle calls are required.  $\square$

### 5.3 $n$ -dimensional subcase with $n_f = 3$

Reconstructing the subdifferential gets more complicated as the number of vertices increases. In this last subcase, we propose a method for the  $n$ -dimensional case where the number of vertices is bounded by  $n_f = 3$ . The proposed strategy exploits the following fact. Let  $Y$  be the projection on a linear subspace of the polytope  $X \subset \mathbb{R}^n$ . Any vertex  $y$  of  $Y$  is the projection of a vertex  $x$  of  $X$  [20]. If the number of vertices is small ( $n_v \in \{1, 2, 3\}$ ), then the combinatorics involved in deducing the vertices of  $X$  from those of  $Y$  is manageable.

We propose the following algorithm that proceeds by successively finding the vertices of the projections of  $X$  on  $\mathbb{R}^2, \mathbb{R}^3, \dots, \mathbb{R}^n$ . The number of vertices of the projection in  $\mathbb{R}^k$  is denoted  $n_v^k$ .

**Algorithm 5.3** Finding  $X_v$  in  $\mathbb{R}^n$ ,

given an oracle  $D(d) = \max_{v \in X} v^\top d$ , and

given the upper bound on the number of vertices is  $n_f = 3 \geq n_v$ .

Initialize:

Apply Algorithm 4.1 to obtain the  $n_v^2$  vertices of the projection of  $X$  in  $\mathbb{R}^2$ :  
 $x^j = (x_1^j, x_2^j, 0, 0, \dots, 0) \in \mathbb{R}^n$  for  $1 \leq j \leq n_v^2$ .

For  $k$  from 3 to  $n$

Given  $x^j = (x_1^j, x_2^j, \dots, x_{k-1}^j, 0, 0, \dots, 0) \in \mathbb{R}^n$  for  $1 \leq j \leq n_v^{k-1}$ .

Call the oracle twice and set  $\ell_k = D(-e_k)$  and  $u_k = D(e_k)$ .

If  $\ell_k = u_k$ , then

set  $n_v^k = n_v^{k-1}$  and reset  $x_k^j \leftarrow \ell_k$  for each  $1 \leq j \leq n_v^k$ .

Otherwise  $\ell_k < u_k$  and apply the appropriate case

• Case I :  $n_v^{k-1} = 1$ .

Set  $n_v^k = 2$ ,  $x^2 = x^1$  and reset  $x_k^1 \leftarrow \ell_k$  and  $x_k^2 \leftarrow u_k$ .

• Case II :  $n_v^{k-1} = 2$ .

The  $n_v^k \leq 3$  vertices of the projection in  $\mathbb{R}^k$  lie in the two-dimensional plane containing  $\{x^i + \ell_k e_k, x^i + u_k e_k : i = 1, 2\}$ .

Use a change of variables to reduce this to a problem in  $\mathbb{R}^2$  and apply Algorithm 4.1 using the current initialization state, as allowed by Lemma 4.6.

• Case III :  $n_v^{k-1} = 3$ .

Set  $n_v^k = 3$ .

For  $j = 1, 2, 3$

Choose  $d \in \mathbb{R}^n$  so that  $d^\top (x^j + \ell_k e_k) = 0$ , and

$$d^\top (x^i + u_k e_k) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \in \{1, 2, 3\}, i \neq j. \end{cases}$$

Call the oracle and reset  $x_k^j \leftarrow \ell_k + D(d)(u_k - \ell_k)$ .

Return  $S_v = \{x^j : 1 \leq j \leq n_v^n\}$ .

**Theorem 5.4 (Convergence of Algorithm 5.3)** Let  $X$  be a polytope in  $\mathbb{R}^n$  with  $n_v \leq n_f = 3$  vertices contained in set  $X_v$ .

If  $n_v = 1$ , then Algorithm 5.3 terminates after at most  $2n - 1$  oracle calls.

If  $n_v = 2$ , then Algorithm 5.3 terminates after at most  $5n - 3$  oracle calls.

If  $n_v = 3$ , then Algorithm 5.3 terminates after at most  $5n - 1$  oracle calls.

In all cases, the algorithm terminates with  $X_v = S_v$ .

**Proof.** The proof is done by induction on  $n$ , the dimension of the space.

If  $n = 2$ , then the algorithm reduces to Algorithm 4.1 and the bounds follow from Theorem 4.3.

Now, suppose that the result is true for some  $n = k - 1 \geq 2$ . Set

$$o_c = \begin{cases} 2n - 1 & \text{if } n_v^n = 1, \\ 5n - 3 & \text{if } n_v^n = 2, \\ 5n - 1 & \text{if } n_v^n = 3, \end{cases} = \begin{cases} 2k - 3 & \text{if } n_v^{k-1} = 1, \\ 5k - 8 & \text{if } n_v^{k-1} = 2, \\ 5k - 6 & \text{if } n_v^{k-1} = 3. \end{cases}$$

Thus, after  $o_c$  oracle calls, the algorithm has correctly identified the  $n_v^{k-1}$  vertices of  $X$  projected onto  $\mathbb{R}^n = \mathbb{R}^{k-1}$  as a subspace of  $\mathbb{R}^k$ .

As the Algorithm 5.3 proceeds into the  $k^{\text{th}}$  iterative loop, it uses 2 oracle calls to compute the bounds  $\ell_k$  and  $u_k$  on the  $k^{\text{th}}$  variable. If both values are equal, then this implies that all vertices belong to the subspace where  $x_k = \ell_k$  and no more calls to the oracle are required, and the algorithm terminates using  $o_c + 2$  oracle calls. If  $\ell_k < u_k$ , then the algorithm proceeds to Case I, II, or III.

In Case I,  $o_c = 2k - 3$  oracle calls sufficed to find the  $n_v^{k-1}$  vertices in the projection on  $\mathbb{R}^{k-1}$ . No other calls are required because all vertices in  $\mathbb{R}^k$  belong to the one dimensional subspace in which the  $k^{\text{th}}$  components lie in the interval  $[x_k + \ell_k, x_k + u_k]$ . The overall number of oracle calls is  $o_c + 2 = 2k - 1$ .

In Case II,  $o_c = 5k - 8$ , and the  $n_v^k$  vertices of the projection in  $\mathbb{R}^k$  must lie in the two dimensional plane containing  $\{x^1 + \ell_k e_k, x^1 + u_k e_k, x^2 + \ell_k e_k, x^2 + u_k e_k\}$ . Remark 4.7 ensures that at most 3 additional oracles calls are required if  $n_v^k = 2$  and at most 5 more oracles calls are required if  $n_v^k = 3$ . So, the algorithm terminates using at most  $o_c + 5 = 5k - 3$  oracle calls if  $n_v = 2$ , and  $o_c + 7 = 5k - 1$  oracle calls if  $n_v = 3$ .

In Case III,  $o_c = 5k - 6$  and  $n_v^k - n_f = 3$ . Figure 11 illustrates Case III. The three vertices must lie on the vertical edges of a triangular prism. One vertex must be on a top-most vertex of this prism, and another vertex must be on a bottom-most vertex of the prism. The final vertex can be located anywhere on the third vertical edge of the prism. The algorithm uses 3 oracle calls to resolve the situation. Figure 11, depicts one step of the inner loop within Case III. The direction  $d$  is selected such that  $d^\top v = 0$  at the three vertices represented by squared, and  $d^\top(x^1 + u_k e_k) = 1$  at the vertex represented by a circle. The hyperplane  $L(d)$  is represented by the shaded region. Calling the oracle  $D(d)$  reveals the position of the vertex on the edge joining  $x^1 + \ell_k e_k$  and  $x^1 + u_k e_k$ . The final result is a total of  $o_c + 5 = 5k - 1$  oracle calls.

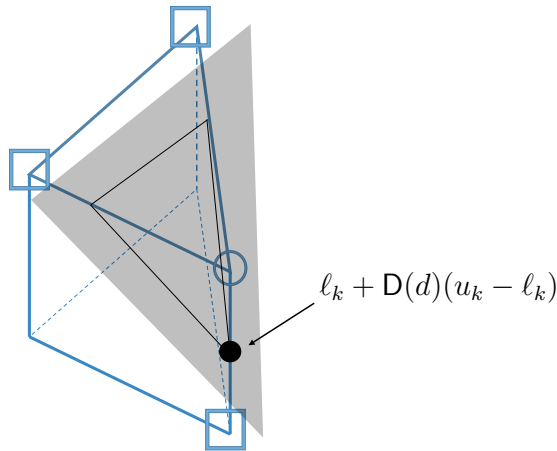
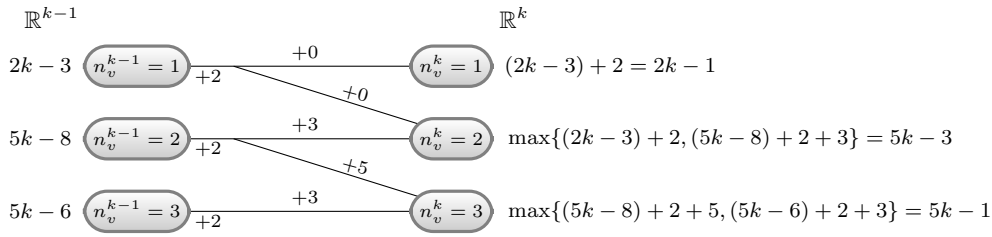


Figure 11: Case III: all three vertices are located on the vertical edges of a triangular prism. The triangular prism is the outer bound created by the initialization stage of Algorithm 5.3. The set  $X$  is the triangle within the prism and the hyperplane  $L(d)$  is represented by the shaded region.

Figure 12 summarizes the total number of calls to the oracle to construct the vertices of the projection in  $\mathbb{R}^k$  from those of  $\mathbb{R}^{k-1}$ . The left part of the figure lists  $o_c$  the number of calls required to identify the  $n_v^{k-1}$  vertices of the projection of  $X$  onto  $\mathbb{R}^{k-1}$ . Immediately after each of the three possible values of  $n_v^{k-1}$  the symbol “+2” indicates that two oracle calls are made to compute  $\ell_k$  and  $u_k$ . The rest of the figure indicates the remaining number of oracle calls.

For example, if  $n_v^k = 3$  then  $n_v^{k-1}$  is necessarily equal to 2 or 3. If  $n_n^{k-1} = 2$  then 5 more calls are necessary (by Case II), and if  $n_n^{k-1} = 3$  then 3 more calls are necessary (by Case III). The maximal value between  $(5k - 8) + 2 + 4$  and  $(5k - 6) + 2 + 3$  constitutes an upper bound on the number of oracle calls when  $n_v^k = 3$ .  $\square$

Figure 12: Number of oracle calls at iteration  $k$  of Algorithm 5.3.

## 6 Discussion

We have studied the question of how to reconstruct a polyhedral subdifferential using directional derivatives. By reformulating the question as the reconstruction of an arbitrary polyhedral  $X$  set using an oracle  $D : \mathbb{R}^n \mapsto \mathbb{R}$  that returns  $D(d) = \max_{v \in X} v^\top d$ , we observed that the question is closely linked to Geometric Probing.

We have developed a number of algorithms that provide methods to reconstruct a polyhedral subdifferential using directional derivatives in various situations. However, many situations remain as open questions. Table 1 summarizes the results in this paper.

Table 1: Maximum number of calls to the oracle to identify  $X_v \subset \mathbb{R}^n$ .

Space	Nb vertices and bound	Nb calls	Source
$\mathbb{R}^1$	$n_f = n_v = 1$	1	Section 3
	$n_f \geq 2$	2	Section 3
$\mathbb{R}^2$	$n_f = n_v = 1$	2	Subsection 5.1
	$n_f > n_v = 1$	3	Algorithm 4.1 with Theorem 4.3
	$n_f = n_v = 2$	5	Algorithm 4.1 with Corollary 4.5
	$n_f = n_v$	$3n_v$	Algorithm 4.1 with Theorem 4.3
	$n_f > n_v$	$3n_v + 1$	Algorithm 4.1 with Theorem 4.3
$\mathbb{R}^n$	$n_f = n_v = 1$	$n$	Subsection 5.1
	$n_f = 2, n_v = 1$	$2n$	Algorithm 5.1 with Theorem 5.2
	$n_f = 2, n_v = 2$	$3n - 1$	Algorithm 5.1 with Theorem 5.2
	$n_f = 3, n_v = 1$	$2n - 1$	Algorithm 5.3 with Theorem 5.4
	$n_f = 3, n_v = 2$	$5n - 3$	Algorithm 5.3 with Theorem 5.4
	$n_f = 3, n_v = 3$	$5n - 1$	Algorithm 5.3 with Theorem 5.4

From Table 1 we see that the problem can be considered solved in  $\mathbb{R}^1$  and  $\mathbb{R}^2$ . However, in  $\mathbb{R}^n$  for an arbitrary bound on the number of vertices  $n_f$  the problem is still open.

This research is inspired in part by recent techniques that create approximate subdifferentials by using a collection of approximate directional derivatives [3, 4, 14]. As such, a natural research direction in this field is to examine how to adapt these algorithms to inexact oracles. That is an oracle  $D^\varepsilon : \mathbb{R}^n \mapsto \mathbb{R}$  that returns  $D^\varepsilon(d) = \max_{v \in X} v^\top d + \xi$ , where  $\xi$  is an unknown error term bounded by  $|\xi| < \varepsilon$ . Some of the algorithms in this paper trivially adapt to this setting. Specifically, the algorithm of Section 3 (for  $\mathbb{R}^1$ ) and the algorithm of Subsection 5.1 (for  $n_f = 1$ ) also work for inexact oracles and the error bounds are trivial to calculate. However, the more interesting algorithms (Algorithms 4.1, 5.1 and 5.3) are not so trivial to adapt.

We conclude this paper with Figure 13, which demonstrates a potential problematic outcome of Algorithm 4.1 if run using an inexact oracle as if it were exact. The continuous lines represent the hyperplane generated by an exact oracle, and the dotted ones are generated by an inexact one. In this example, the fifth oracle call is incompatible with Algorithm 4.1, as no new vertices are discovered.

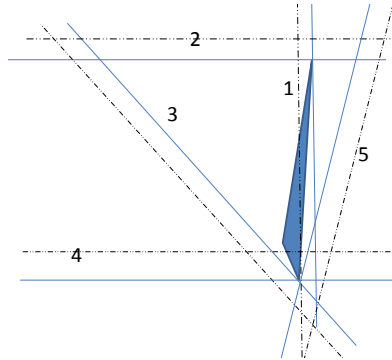


Figure 13: Possible outcome of Algorithm 4.1 when run with an inexact oracle. After 5 oracle calls the algorithm stalls, as no new vertices are discovered.

## References

- [1] C. Audet and J.E. Dennis, Jr. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 13(3):889–903, 2003.
- [2] C. Audet and J.E. Dennis, Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006.
- [3] A.M. Bagirov. Continuous subdifferential approximations and their applications. *Journal of Mathematical Sciences (New York)*, 115(5):2567–2609, 2003. Optimization and related topics, 2.
- [4] A.M. Bagirov, B. Karasözen, and M. Sezer. Discrete gradient method: derivative-free method for nonsmooth optimization. *Journal of Optimization Theory and Applications*, 137(2):317–334, 2008.
- [5] H.J. Bernstein. Determining the shape of a convex  $n$ -sided polygon by using  $2n + k$  tactile probes. *Information Processing Letters*, 22(5):255–260, 1986.
- [6] R. Cole and C.K. Yap. Shape from probing. *Journal of Algorithms*, 8(1):19–38, 1987.
- [7] D. Dobkin, H. Edelsbrunner, and C.K. Yap. Probing convex polytopes. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, STOC '86*, pages 424–432, New York, NY, USA, 1986. ACM.
- [8] D.E. Finkel and C.T. Kelley. Convergence analysis of the DIRECT algorithm. Technical Report CRSC-TR04-28, Center for Research in Scientific Computation, 2004.
- [9] D.E. Finkel and C.T. Kelley. Convergence analysis of sampling methods for perturbed lipschitz functions. *Pacific Journal of Optimization*, 5(2):339–350, 2009.
- [10] W. Hare and J. Nutini. A derivative-free approximate gradient sampling algorithm for finite minimax problems. *Computational Optimization and Applications*, 56(1):1–38, 2013.
- [11] W. Hare and M. Macklem. Derivative-free optimization methods for finite minimax problems. *Optimization Methods and Software*, iFirst:1–13, 2011.
- [12] J.B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms. II*, volume 306 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1993. Advanced theory and bundle methods.
- [13] A. Imiya and K. Sato. Shape from silhouettes in discrete space. In E.V. Brimkov and P.R. Barneva, editors, *Digital Geometry Algorithms: Theoretical Foundations and Applications to Computational Imaging*, pages 323–358. Springer Netherlands, Dordrecht, 2012.
- [14] K.C. Kiwiel. A nonderivative version of the gradient sampling algorithm for nonsmooth nonconvex optimization. *SIAM Journal on Optimization*, 20(4):1983–1994, 2010.
- [15] M. Lindenbaum and A. Bruckstein. Parallel strategies for geometric probing. *Journal of Algorithms*, 13(2):320–349, 1992.
- [16] R.T. Rockafellar and R.J.-B. Wets. *Variational Analysis*, volume 317 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1998.
- [17] K. Romanik. Geometric probing and testing – a survey. Technical Report 95–42, DIMACS Technical Report, 1995.
- [18] R.L. Shuo-Yen. Reconstruction of polygons from projections. *Information Processing Letters*, 28(5):235–240, 1988.

- [19] S.S. Skiena. Problems in geometric probing. *Algorithmica*, 4(1):599–605, 1989.
- [20] H.R. Tiwary. Complexity of some polyhedral enumeration problems. PhD thesis, Universität des Saarlandes, Germany, 2008.