Maximizing Concave Functions in Fixed Dimension

Edith Cohen[1]
AT&T Bell Laboratories, Murray Hill, NJ 07928 USA

Nimrod Megiddo
IBM Research, Almaden Research Center, San Jose, CA 95120-6099 USA, and School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel.

Abstract

In [3, 7, 2] the authors introduced a technique which enabled them to solve the parametric minimum cycle problem with a fixed number of parameters in strongly polynomial time. In the current paper[2] we present this technique as a general tool. In order to allow for an independent reading of this paper, we repeat some of the definitions and propositions given in [3, 7, 2]. Some proofs are not repeated, however, and instead we supply the interested reader with appropriate pointers.

Suppose $\mathcal{Q} \subset R^d$ is a convex set given as an intersection of k halfspaces, and let $g : \mathcal{Q} \rightarrow R$ be a concave function that is computable by a piecewise affine algorithm (i.e., roughly, an algorithm that performs only multiplications by scalars, additions, and comparisons of intermediate values which depend on the input). Assume that such an algorithm $\mathcal{A}$ is given and the maximal number of operations required by $\mathcal{A}$ on any input (i.e., point in $\mathcal{Q}$) is T. We show that under these assumptions, for any fixed d, the function g can be maximized in a number of operations polynomial in k and T. We also present a general framework for parametric extensions of problems where this technique can be used to obtain strongly polynomial algorithms. Norton, Plotkin, and Tardos [13] applied a similar scheme and presented additional applications.

[1] Research done while the author was at Stanford University and IBM Almaden Research Center. Research partially supported by NSF PYI Grant CCR-8858097, and ONR Contract N00014-88-K-0166.

[2] See [4, 2] for an earlier version.

1 Introduction

A *convex optimization problem* is a problem of maximizing a concave function g over a convex set $S \subset R^d$. Equivalently, we can consider minimizing a convex function. We consider the problem of maximizing a concave function g, where the dimension of the space is fixed. We assume that g is provided to us by a piecewise affine algorithm (see Definition 2.2) that can evaluate g at any given point.

The results of this paper can be extended easily to the case where the range of g is R^ℓ for any $\ell \geq 1$. We then define the notions of maximum and concavity of g with respect to the lexicographic order as follows. We say that a function $g : \mathcal{Q} \subset R^d \to R^\ell$ is *concave* with respect to the lexicographic order $\leq_{\text{lex}}$ if for every $\alpha \in [0,1]$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{Q}$,

$$\alpha g(\boldsymbol{x}) + (1-\alpha) g(\boldsymbol{y}) \leq_{\text{lex}} g(\alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{y}) \; .$$

Applications where the range of g is R^2 were given in [5, 8].

In Section 2 we define the problem. In Section 3 we introduce the subproblem of hyperplane queries, which is essential for the design of our algorithm. In Section 4 we discuss the multi-dimensional search technique which we utilize for improving our time bounds. In Section 5 we introduce the optimization algorithm. In Sections 6 and 7 we prove the correctness and analyze the time complexity of the algorithm. In Section 8 we discuss applications of the technique introduced here to obtain strongly polynomial time algorithms for parametric extensions of other problems.

2 Preliminaries

Let R_1^d denote the set of vectors $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_d)^T \in R^d$ such that $\lambda_d = 1$. For $\boldsymbol{\lambda} \in R^{d-1}$, denote by $\hat{\boldsymbol{\lambda}} \in R_1^d$ the vector $\hat{\boldsymbol{\lambda}} = (\boldsymbol{\lambda}, 1)$. For a halfspace F, denote by ∂F the boundary of F, i.e., ∂F is a hyperplane.

Definition 2.1 For a finite set $\mathcal{C} \subset R^{d+1}$, denote by $L_{\mathcal{C}} : R^{d+1} \to R$ the minimum envelope of the linear functions that correspond to the vectors in $\mathcal{C}$, i.e.,

$$L_{\mathcal{C}}(\boldsymbol{\lambda}) = \min_{\boldsymbol{c} \in \mathcal{C}} \boldsymbol{c}^T \boldsymbol{\lambda} \; .$$

Denote by $\hat{L}_{\mathcal{C}} : R^d \to R$ the function given by

$$\hat{L}_{\mathcal{C}}(\boldsymbol{\lambda}) = L_{\mathcal{C}}(\hat{\boldsymbol{\lambda}}) \; .$$

The vectors in $\mathcal{C} \subset R^{d+1}$ are called the *pieces of* $\hat{L}_{\mathcal{C}}$. For a piece $\boldsymbol{c} \in \mathcal{C}$ and a vector $\boldsymbol{\beta} \in R^d$ such that $\boldsymbol{c}^T \hat{\boldsymbol{\beta}} = \hat{L}_{\mathcal{C}}(\boldsymbol{\beta})$, we say that $\boldsymbol{c}$ is *active* at $\boldsymbol{\beta}$.

Definition 2.2 For a function $g : \mathcal{Q} \to R$, where $\mathcal{Q} \subset R^d$:

i. Denote by Λ_g (or Λ, for brevity) the set of maximizers of g. The set Λ may be empty.

ii. An algorithm that computes the function g (i.e., for $\boldsymbol{\lambda} \in \mathcal{Q}$ returns the value $g(\boldsymbol{\lambda})$ and otherwise stops or returns an arbitrary value) is called *piecewise affine*, if the operations it performs on intermediate values that depend on the input vector are restricted to additions, multiplications by constants, comparisons, and copies.

iii. For a piecewise affine algorithm $\mathcal{A}$, denote by $T(\mathcal{A})$ and $C(\mathcal{A})$ the maximum number of operations and the maximum number of comparisons, respectively, performed by $\mathcal{A}$. We assume that this numbers are finite.

iv. If $g = \hat{L}_{\mathcal{C}}$ for some $\mathcal{C} \subset R^{d+1}$, we say that $g' = \hat{L}_{\mathcal{C}'}$ is a *weak approximation* of g, if the set of pieces of g' is a subset of the set of pieces of g ($\mathcal{C}' \subset \mathcal{C}$), and the affine hulls[3] aff Λ_g and aff $\Lambda_{g'}$ are equal. The function $g' = \hat{L}_{\mathcal{C}'}$ is a *minimal* weak approximation of g, if there is no $\mathcal{C}'' \subset C'$ such that $\hat{L}_{\mathcal{C}''}$ is a weak approximation of g.

Remark 2.3 Suppose that $\mathcal{A}$ is a piecewise affine algorithm. Consider the computation tree (i.e., the tree consisting of all possible computation paths) of $\mathcal{A}$. Observe that all the intermediate values along a computation path, including the final output, can be expressed as linear functions (i.e., are of the form $\boldsymbol{a}^T\hat{\boldsymbol{\lambda}}$) of the input vector. These linear functions can be easily computed and maintained during a single execution of the algorithm. These linear functions map the input vectors whose computation path coincides so far with that same path to the corresponding value. Moreover, the linear function which corresponds to the final output at a single execution is a piece which is active at the input vector.

Remark 2.4 Suppose $g : \mathcal{Q} \subset R^d \to R$ is concave and computable by a piecewise affine algorithm $\mathcal{A}$. It is easy to see that there exists a finite set $\mathcal{C} \subset R^{d+1}$ such that g coincides with $\hat{L}_{\mathcal{C}}$.

Definition 2.5 Suppose $\mathcal{Q} = F_1 \cap \cdots \cap F_k$ is the intersection of k closed halfspaces and $g : \mathcal{Q} \to R$, $g = \hat{L}_{\mathcal{C}}$, is concave and computable by a piecewise affine algorithm.

i. For $\boldsymbol{\beta} \in R^d$, denote

$$\mathcal{Q}_\beta = \bigcap \{F_i \ : \ \boldsymbol{\beta} \in \partial F_i\} \ .$$

Denote by $g_\beta : \mathcal{Q}_\beta \subset R^d \to R$ the function whose pieces are all the pieces of g which are active at $\boldsymbol{\beta}$,

$$g_\beta(\boldsymbol{\lambda}) = \min_{c \in \mathcal{C}} \{\boldsymbol{c}^T\hat{\boldsymbol{\lambda}} \ : \ \boldsymbol{c}^T\hat{\boldsymbol{\beta}} = g(\boldsymbol{\beta})\} \ .$$

Note that $g_\beta = \hat{L}_{\mathcal{C}'}$ where $\mathcal{C}' = \{\boldsymbol{c} \in \mathcal{C} \mid \boldsymbol{c}^T\hat{\boldsymbol{\beta}} = g(\boldsymbol{\beta})\}$. See Figure 1 for an example. Later, we describe an algorithm for evaluating g_β.

[3]The affine hull aff S of a set S is the smallest flat that contains S.

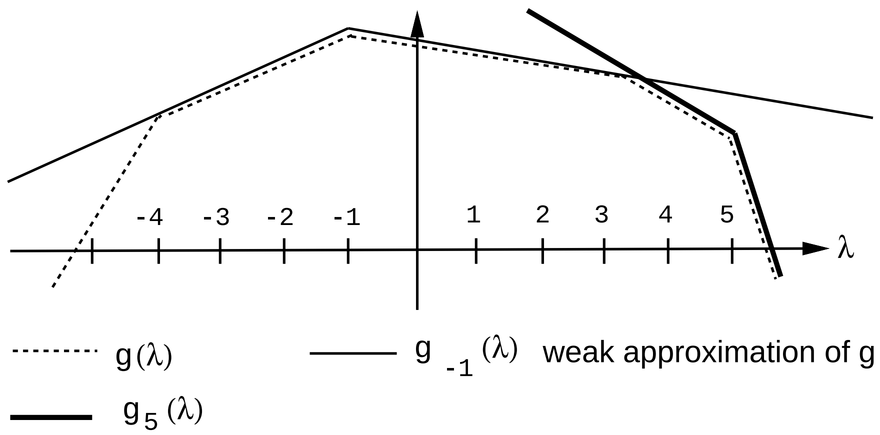

Figure 1: Examples of restrictions of g

ii. Suppose an ℓ-dimensional flat $S \subset R^d$, is represented as a set of solutions of a linear system of equations. There exists an affine mapping M from R^ℓ onto S, which can be computed in $O(d^3)$ time. Denote $\mathcal{Q}^S = \{\boldsymbol{\lambda} \in R^\ell \mid M(\boldsymbol{\lambda}) \in \mathcal{Q}\}$, and define $g^S : \mathcal{Q}^S \to R$ by $g^S = g \circ M$.

Proposition 2.6 *Let $\mathcal{A}$ be a piecewise affine algorithm for evaluating $g : \mathcal{Q} \to R$, where $\mathcal{Q} \subset R^d$ is given as the intersection of k halfspaces. By modifying $\mathcal{A}$ we can obtain the following piecewise affine algorithms:*

i. For any given vector $\boldsymbol{\beta} \in \mathcal{Q}$, we obtain an algorithm $\mathcal{A}_\beta$ for evaluating g_β so that $T(\mathcal{A}_\beta) = T(\mathcal{A}) + dC(\mathcal{A})$ and $C(\mathcal{A}_\beta) = C(\mathcal{A})$.

ii. For any ℓ-dimensional flat $S \subset R^d$, represented as the set of solutions of a system of linear equations, we obtain an algorithm $\mathcal{A}^S$ for evaluating g^S so that $T(\mathcal{A}^S) = T(\mathcal{A}) + O(\ell d)$ and $C(\mathcal{A}^S) = C(\mathcal{A})$.

Proof: Part ii is straightforward, since we can choose the algorithm $\mathcal{A}^S$ to be a composition of the appropriate affine mapping and $\mathcal{A}$. We discuss the construction of the algorithm $\mathcal{A}_\beta$ for part i. Consider an input vector $\boldsymbol{\lambda} \in \mathcal{Q}_\beta$. Let $\epsilon > 0$ be such that for all ϵ' ($0 < \epsilon' \leq \epsilon$), $\boldsymbol{\beta} + \epsilon'(\boldsymbol{\lambda} - \boldsymbol{\beta}) \in \mathcal{Q}$, and the set of pieces of g which are active at $\boldsymbol{\beta} + \epsilon'(\boldsymbol{\lambda} - \boldsymbol{\beta})$ is equal to the set of pieces which are active at $\boldsymbol{\beta} + \epsilon(\boldsymbol{\lambda} - \boldsymbol{\beta})$. It is immediate to see that such an ϵ always exists. It follows from the definition that $g_\beta(\boldsymbol{\lambda})$ is the value of $\boldsymbol{\lambda}$ at the linear pieces of g which are active at $\boldsymbol{\beta} + \epsilon(\boldsymbol{\lambda} - \boldsymbol{\beta})$. The algorithm $\mathcal{A}_\beta$, when executed with an input $\boldsymbol{\lambda}$, follows the computation path of $\mathcal{A}$ which corresponds to the input $\boldsymbol{\beta}+\epsilon(\boldsymbol{\lambda}-\boldsymbol{\beta})$. The algorithm computes the linear functions associated with the intermediate values of this path (see Remark 2.3). Recall that the linear function which corresponds to the final value is a piece of g which is active at $\boldsymbol{\beta} + \epsilon(\boldsymbol{\lambda} - \boldsymbol{\beta})$. Hence, the value of $g_\beta(\boldsymbol{\lambda})$ is obtained by substituting $\boldsymbol{\lambda}$ in this linear function. In order to follow the desired run of $\mathcal{A}$, the algorithm $\mathcal{A}_\beta$ mimics the work of $\mathcal{A}$ on additions and multiplications by scalars, keeping track of linear functions rather than just numerical values. When the run of $\mathcal{A}$ reaches a comparison (branching point), $\mathcal{A}_\beta$ does as follows. Without loss of generality we assume that the branching is according to the sign of the linear function $\boldsymbol{a}^T\hat{\boldsymbol{x}}$. In order to decide what to do at a branching point, $\mathcal{A}_\beta$ has to determine the sign of $\boldsymbol{a}^T\hat{\boldsymbol{x}}$ at the point $\boldsymbol{x} = \boldsymbol{\beta} + \epsilon(\boldsymbol{\lambda} - \boldsymbol{\beta})$. Since ϵ is not given explicitly, the decision cannot be made directly by substitution. The decision is made as follows. The algorithm first computes $\alpha = \boldsymbol{a}^T\hat{\boldsymbol{\beta}}$. If $\alpha \neq 0$, then obviously for any vector $\boldsymbol{y}$, for sufficiently small number $\epsilon > 0$ $\boldsymbol{a}^T(\widehat{\boldsymbol{\beta} + \epsilon\boldsymbol{y}})$ has the same sign as α. In particular this holds for $\boldsymbol{y} = \boldsymbol{\lambda} - \boldsymbol{\beta}$ and the sign is detected. Otherwise, if $\alpha = 0$, it follows that $\boldsymbol{a}^T(\widehat{\boldsymbol{\beta} + \epsilon(\boldsymbol{\lambda} - \boldsymbol{\beta})}) = \epsilon\boldsymbol{a}^T\hat{\boldsymbol{\lambda}}$. Hence $\boldsymbol{a}^T\hat{\boldsymbol{\lambda}}$ has the same sign as $\boldsymbol{a}^T(\widehat{\boldsymbol{\beta} + \epsilon(\boldsymbol{\lambda} - \boldsymbol{\beta})})$. It remains to compute the sign of $\boldsymbol{a}^T\hat{\boldsymbol{\lambda}}$ and branch accordingly. It is easy to verify that $\mathcal{A}_\beta$ evaluates the function g_β for any vector $\boldsymbol{\lambda} \in \mathcal{Q}_\beta$, and performs the stated number of operations. ∎

Proposition 2.7 *If $\boldsymbol{\beta} \in \Lambda_g$ then g_β is a weak approximation of g (see [7, 2]) for a proof).*

The goal is to solve the following problem:

Problem 2.8 The input of this problem consists of a polyhedron $\mathcal{Q} = F_1 \cap \cdots \cap F_k$, given as the intersection of k closed halfspaces and a piecewise affine algorithm $\mathcal{A}$ for evaluating a concave function $g : \mathcal{Q} \to R$. Decide whether or not g is bounded. If so, then find a $\boldsymbol{\lambda}^* \in \operatorname{rel\,int} \Lambda$. We refer to the following as the "optional" part of the problem: If g is bounded, then find a subset $\mathcal{C}$ of the set of pieces of g, such that $\hat{L}_\mathcal{C}$ is a minimal weak approximation of g, and $|\mathcal{C}| \leq 2d$.

The set $\mathcal{C}$ may be viewed as a *certificate* for the fact that the maximum of the function g does not exceed $g(\boldsymbol{\lambda}^*)$. In the current paper we do not discuss the details of solving the optional part of the problem. See [7, 2] for an existence proof and an algorithm which finds such a set.

We propose an algorithm for Problem 2.8. In any fixed dimension d, the total number of operations performed by this algorithm is bounded by a polynomial in $T(\mathcal{A})$ and k. The algorithm is based on solving instances of a subproblem, which we call *hyperplane query*: For a given hyperplane H_0, decide on which side of H_0 the function g is either unbounded or attains its maximum. A procedure for hyperplane queries is called an *oracle*. Obviously, an oracle can be utilized to perform a binary search over the polyhedron $\mathcal{Q}$. However, in order to attain an exact solution within time bounds that depend only on d, T, and k, we use the oracle in a more sophisticated way. The number of hyperplane queries needed by the algorithm, and hence the number of oracle calls, is bounded by the number of comparisons performed by $\mathcal{A}$.

We later discuss applying the multi-dimensional search technique, what allows us to do even better. When $\mathcal{A}$ is a parallel algorithm, we may reduce the number of queries corresponding to a single parallel step to a polylogarithm of the number of concurrently performed comparisons.

The function g is a concave piecewise linear mapping. Concave functions have the property that it can be effectively decided which side of a given hyperplane H_0 contains the maximum of the function. If the domain of g does not intersect H_0, then the answer is the side of H_0 which contains the domain of g. Otherwise, the decision can be made by considering a neighborhood of the maximum of the function relative to H_0, searching for a direction of ascent from that point. This principle is explained in detail in [12].

For a hyperplane $H_0 \subset R^d$, we wish to decide on which side of H_0 the set $\operatorname{rel\,int} \Lambda$ lies. By solving a linear program with d variables and $k+1$ constraints, we determine whether or not $H_0 \cap \mathcal{Q} = \emptyset$, and if so, we determine which side of H_0 contains $\mathcal{Q}$. It follows from [12] that this can be done in $O(k)$ time. If $H_0 \cap \mathcal{Q} \neq \emptyset$, then the oracle problem solves the original problem, when g is restricted to H_0. If g is unbounded on H_0 the oracle reveals that. If $\Lambda = \emptyset$, or if $\operatorname{rel\,int} \Lambda$ is either contained in H_0 or extends into both sides of H_0 (i.e., $H_0 \cap \operatorname{rel\,int} \Lambda \neq \emptyset$), then we find $\boldsymbol{\lambda} \in H_0 \cap \operatorname{rel\,int} \Lambda$ and the oracle will actually solve Problem 2.8.

Problem 2.9 Given are a set $\mathcal{Q} = F_1 \cap \cdots \cap F_k$, a piecewise affine algorithm $\mathcal{A}$ which evaluates a concave function $g : \mathcal{Q} \to R$, and a hyperplane H_0 in R^d. Do as follows:

i. If $\mathcal{Q} \cap H_0 = \emptyset$, recognize which of the two halfspaces determined by H_0 contains $\mathcal{Q}$. Otherwise,

ii. recognize whether or not g is bounded on H_0. If it is, then

iii. find $\boldsymbol{\lambda} \in H_0 \cap \operatorname{rel\,int} \Lambda$ if such $\boldsymbol{\lambda}$ exists, and solve Problem 2.8 relative to g. Otherwise, if $H_0 \cap \operatorname{rel\,int} \Lambda = \emptyset$, then

iv. recognize which of the two halfspaces determined by H_0 has either a nonempty intersection with $\operatorname{rel\,int} \Lambda$, or has g unbounded on it.

A procedure for solving Problem 2.9 will be called an *oracle* and the hyperplane H_0 will be called the *query hyperplane*. Problem 2.8 is solved by running a modification of the algorithm $\mathcal{A}$, where additions and multiplications are replaced by vector operations and comparisons are replaced by hyperplane queries. Problem 2.9 is solved by three recursive calls to instances of Problem 2.8 of the form $(\mathcal{A}^H, \mathcal{Q}^H)$, $(\mathcal{A}^H_\beta, \mathcal{Q}^H_\beta)$, where $\boldsymbol{\beta} \in R^d$, and H is a hyperplane (see Definitions 2.5 and 2.6). Note that these algorithms compute, respectively, the functions $g^H : \mathcal{Q}^H \to R$, $g^H_\beta : \mathcal{Q}^H_\beta \to R$, where $\mathcal{Q}^H$ and $\mathcal{Q}^H_\beta$ are subsets of R^{d-1}. Hence, the recursive calls are made to instances of lower dimension.

In Section 5 we propose Algorithm 5.2 for Problem 2.8. The algorithm executes calls to the oracle problem (Problem 2.9) relative to g. An algorithm for the oracle problem is given in Section 3. A call to the oracle is costly. Therefore, one wishes to solve many hyperplane queries with a small number of oracle calls. In Section 4 we discuss the multi-dimensional search technique (introduced in [12] and later refined in [1, 9]).

3 Hyperplane queries

For a hyperplane $H \subset R^d$, we solve Problem 2.9 for g relative to H.

Theorem 3.1 *Problem 2.9 can be reduced to the problem of solving three instances of Problem 2.8 on functions defined on an intersection of at most k closed halfspaces in R^{d-1}. The time complexity of the additional computation is $O(d^3)$.*

Proof: We solve Problem 2.8 with the function g^H, where $H = \{\boldsymbol{\lambda} \in R^d \mid \boldsymbol{a}^T\hat{\boldsymbol{\lambda}} = \alpha\}$. If g is unbounded on H, then this fact is detected; otherwise, suppose $\boldsymbol{\lambda}^{(0)}$ is in the relative interior of the set of maximizers of $g(\boldsymbol{\lambda})$ subject to $\boldsymbol{\lambda} \in H$, and we get the collection $\mathcal{C}^{(0)}$. Let $t^{(0)} = g(\boldsymbol{\lambda}^{(0)})$. We wish to recognize whether $\boldsymbol{\lambda}^{(0)}$ is also a relative interior point of the set of global maxima (i.e., relative to R^d). If not, then we wish to decide whether for all $\boldsymbol{\lambda}^* \in R^d$ such that $g(\boldsymbol{\lambda}^*) \geq g(\boldsymbol{\lambda}^{(0)})$, necessarily $\boldsymbol{a}^T\boldsymbol{\lambda}^{*\prime} > \alpha$, or whether for all of them $\boldsymbol{a}^T\boldsymbol{\lambda}^{*\prime} < \alpha$. These are the two possible cases. Consider the function $g_{\lambda^{(0)}}$. We solve Problem 2.8 on two restrictions of $g_{\lambda^{(0)}}$ to hyperplanes (see Proposition 2.6), where in one case it is restricted to $H^{(1)} = \{\boldsymbol{\lambda} \mid \boldsymbol{a}^T\hat{\boldsymbol{\lambda}} = \alpha - 1\}$, and in the other to $H^{(-1)} = \{\boldsymbol{\lambda} \mid \boldsymbol{a}^T\hat{\boldsymbol{\lambda}} = \alpha + 1\}$. Note that the domains $\mathcal{Q}_{\lambda^{(0)}}^{H^{(\delta)}}$ $(\delta \in \{-1, 1\})$ are $(d-1)$-dimensional. Denote the respective optimal values of $g_{\lambda^{(0)}}^{H^{(\delta)}}$ by $t^{(\delta)}$ $(\delta \in \{-1, 1\})$, and let $\mathcal{C}^\delta$ be the respective minimal weak approximations. Only one of the optimal values $t^{(1)}, t^{(-1)}$ can be greater than $t^{(0)}$. If this is the case, or if one of $t^{(1)}, t^{(-1)}$ equals $t^{(0)}$ and the other is smaller, then the side of the hyperplane that contains $\operatorname{rel\,int}\Lambda$ is determined. Otherwise, if both values are less than or both values are equal to $t^{(0)}$, then $t^{(0)}$ is the global optimal value. In the latter case $\boldsymbol{\lambda}^{(0)} \in \operatorname{rel\,int}\Lambda$. It follows from Proposition 2.7 that the pieces of g active in a minimal weak approximation have the value $t^{(0)}$ at $\boldsymbol{\lambda}^{(0)}$. Thus, a minimal weak approximation of the function $g_{\lambda^{(0)}}$ is a minimal weak approximation of g. It follows from analysis done in [7, 2] that by using $O(d^3)$ operations we can construct a minimal weak approximation of $g_{\lambda^{(0)}}$. Furthermore, the number of pieces involved in a minimal weak approximation is at most $2d$. ∎

As an example, consider an application of the algorithm described in the proof to decide on which side of the hyperplane $H = \{2\}$ the function $g(\boldsymbol{\lambda}) = \min\{\boldsymbol{\lambda}/5 + 2, -4\boldsymbol{\lambda} + 12.5\}$ is maximized (see Figure 2). Note that maximizing a function $f : R \to R$ on a hyperplane corresponds to evaluating it at a single point. Therefore, the maximum value of g on H is 2.4. The algorithm considers the restriction $g_2 = \boldsymbol{\lambda}/5 + 2$,

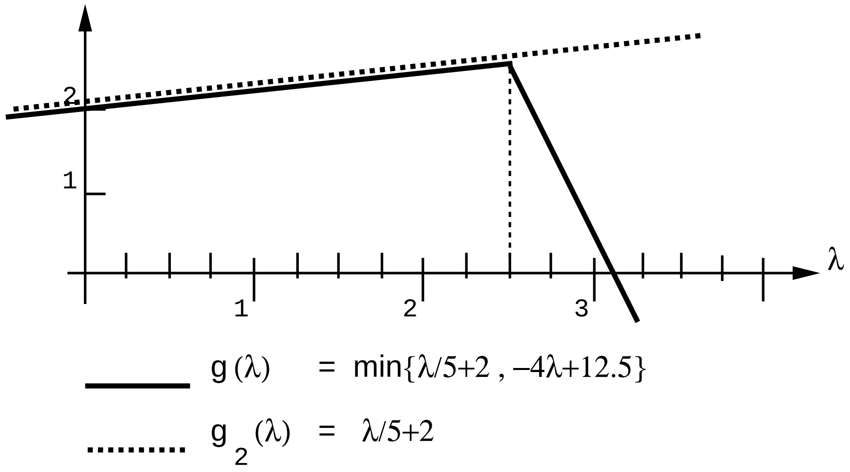

Figure 2: Example: hyperplane query at $H = \{2\}$

and maximizes it on the hyperplanes $H^{(1)} = \{1\}$ and $H^{(-1)} = \{3\}$. The corresponding maxima are $t^{(1)} = 2.2$ and $t^{(-1)} = 2.6$, and hence, the algorithm concludes that the maximizers of g are contained in the halfspace $\{\boldsymbol{\lambda} \in R | \boldsymbol{\lambda} > 2\}$. Observe that this conclusion could not have been made if the algorithm considered the values of g, rather than the values of the restriction g_2, at the hyperplanes $\{1\}$ and $\{3\}$.

4 Employing multi-dimensional search

The definitions and propositions stated in this section appeared in [3, 7, 2]. They are presented here to allow for an independent reading of this paper. For proofs, the reader is referred to [3, 7, 2]. The multi-dimensional search problem was defined and used in [12] for solving linear programming problems in fixed dimension. In this section we employ it to achieve better time bounds.

Definition 4.1 We define a partial order on $R^d \setminus \{\mathbf{0}\}$, relative to a concave function $g : \mathcal{Q} \to R$, where $\mathcal{Q} \subset R^{d-1}$ is a nonempty polyhedral set. For any pair of distinct vectors $\boldsymbol{a}_1, \boldsymbol{a}_2 \in R^d$, denote

$$H = H(\boldsymbol{a}_1, \boldsymbol{a}_2) = \{\boldsymbol{\lambda} \in R^{d-1} : \boldsymbol{a}_1^T \hat{\boldsymbol{\lambda}} = \boldsymbol{a}_2^T \hat{\boldsymbol{\lambda}}\} \ .$$

If g is unbounded on $H(\boldsymbol{a}_1, \boldsymbol{a}_2)$ or if $H(\boldsymbol{a}_1, \boldsymbol{a}_2) \cap \operatorname{rel\,int} \Lambda \neq \emptyset$, then we write $\boldsymbol{a}_1 <>_\Lambda \boldsymbol{a}_2$. Otherwise, g can be unbounded on at most one of the open halfspaces determined by H, and also $\operatorname{rel\,int} \Lambda$ can intersect at most one of these open halfspaces. If g is undefined on H (i.e., $\mathcal{Q} \cap H = \emptyset$), then $\mathcal{Q}$ is contained in one of these halfspaces. We denote $\boldsymbol{a}_1 <_\Lambda \boldsymbol{a}_2$ (respectively, $\boldsymbol{a}_1 >_\Lambda \boldsymbol{a}_2$) if there exists a $\hat{\boldsymbol{\lambda}} \in \operatorname{rel\,int} \Lambda$ such that $\boldsymbol{a}_1^T \hat{\boldsymbol{\lambda}} < \boldsymbol{a}_2^T \hat{\boldsymbol{\lambda}}$ (respectively, $\boldsymbol{a}_1^T \hat{\boldsymbol{\lambda}} > \boldsymbol{a}_2^T \hat{\boldsymbol{\lambda}}$), in which case the same holds for all these $\boldsymbol{\lambda}$'s, or if g is unbounded on the halfspace determined by the inequality $\boldsymbol{a}_1^T \hat{\boldsymbol{\lambda}} < \boldsymbol{a}_2^T \hat{\boldsymbol{\lambda}}$

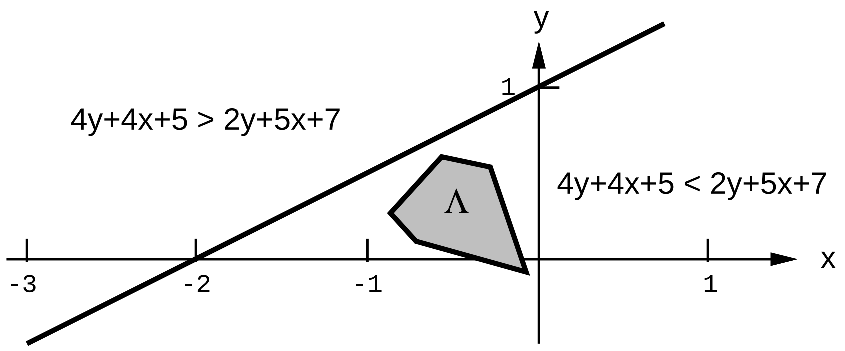

Figure 3: An example where $(4,4,5) <_\Lambda (2,5,7)$

(respectively, $\boldsymbol{a}_1^T\hat{\boldsymbol{\lambda}} < \boldsymbol{a}_2^T\hat{\boldsymbol{\lambda}}$). See Figure 3 for an example. We also use the notation $<_P$ for a similar partial order relative to any set P.

Problem 4.2 Suppose we are given finite sets $A_1, \ldots, A_r$ of nonzero vectors, where $A_i = \{\boldsymbol{a}_1^i, \ldots, \boldsymbol{a}_{s_i}^i\}$ ($\boldsymbol{a}_j^i \in R^d$) and $s = \sum s_i$. The goal is either to find a minimal element, with respect to the partial order $<_\Lambda$, in each of the sets A_i, or (if we encounter two incomparable elements) to reduce the problem to a lower dimension. More specifically, we need to do either one of the following:

i. Find a collection of closed halfspaces whose intersection P contains $\operatorname{rel\,int}\Lambda$, and indices $1 \le m_i \le s_i$ ($i = 1, \ldots, r$) as follows. For every $1 \le i \le r$ and every $1 \le j \le s_i$, $j \ne m_i$, we have $\boldsymbol{a}_{m_i}^i <_\Lambda \boldsymbol{a}_j^i$ and $\boldsymbol{a}_{m_i}^i <_P \boldsymbol{a}_j^i$.

ii. Find a hyperplane H such that either g is unbounded on H or $H \cap \operatorname{rel\,int}\Lambda \ne \emptyset$.

Proposition 4.3 *Problem 4.2 can be solved using $O(\gamma(d-1)\log s)$ oracle calls plus additional computation which can be performed in either*

i. $O(\gamma(d-1)\log^2 s)$ parallel time on $O(s)$ processors, or

ii. $O(\gamma(d-1)s\log s)$ sequential time.

The function $\gamma(d)$ arises from the multi-dimensional search [12]. It follows from [1, 9] that $\gamma(d) = 3^{O(d^2)}$.

5 The algorithm

The algorithm described below solves Problem 2.8. It finds a vector $\boldsymbol{\lambda}^* \in \operatorname{rel\,int}\Lambda$, unless g is unbounded. It also returns a collection $\mathcal{C}$ of pieces of g whose minimum envelope $\hat{L}_\mathcal{C}$ is a minimal weak approximation of g. The number of vectors in $\mathcal{C}$ is at most $2d$.

Definition 5.1 For a piecewise affine algorithm $\mathcal{A}$, we define the corresponding *lifted computation*. The lifted computation is a run of the algorithm on a set of inputs. The computation is done symbolically on linear functions instead of on scalars. It follows the path on the computation tree of $\mathcal{A}$ that corresponds to input vectors which are in Λ. The additions and scalar multiplications are trivially generalized to operations on linear functions. When a comparison is done between $\boldsymbol{f}_1^T\hat{\boldsymbol{\lambda}}$ and $\boldsymbol{f}_2^T\hat{\boldsymbol{\lambda}}$, it is resolved according to the partial order $<_\Lambda$. We compute the hyperplane $H(\boldsymbol{f}_1, \boldsymbol{f}_2)$ and solve Problem 2.9 (hyperplane query) relative to H. The hyperplane query decides whether or not the vectors are comparable. If they are, it decides whether $\boldsymbol{f}_1 <_\Lambda \boldsymbol{f}_2$. If $\boldsymbol{f}_1 <>_\Lambda \boldsymbol{f}_2$, then the lifted computation halts since an oracle call resulted in a solution to Problem 2.8. Otherwise, the resolved hyperplane query tells us which of the halfspaces defined by $H(\boldsymbol{f}_1, \boldsymbol{f}_2)$ contains the set $\operatorname{rel\,int}\Lambda$, and the comparison is resolved.

Sets of independent comparisons performed by $\mathcal{A}$ correspond to sets of independent hyperplane queries. Recall from Section 4 that a set of independent hyperplane queries can be solved by performing a logarithmic number of "oracle" calls. The lifted computation maintains a set $\mathcal{H}$ of closed halfspaces which is initially empty. Whenever an oracle call is executed the resulting halfspace is added to $\mathcal{H}$.

Algorithm 5.2 [Find a vector $\boldsymbol{\lambda} \in \operatorname{rel\,int}\Lambda_g$]

Step 1. Run the lifted computation, collecting into $\mathcal{H}$ all the halfspaces resulting from oracle calls where comparisons are resolved. If the computation halts, then some comparison is not resolved but a global solution is found, so stop. Otherwise, denote by $\boldsymbol{m} = (m_1, \ldots, m_{d+1})^T \in R^{d+1}$ the piece of g that corresponds to the computation path followed.

Step 2. Denote by P the intersection of the halfspaces in $\mathcal{H}$.

i. Compute $\boldsymbol{\lambda}^* \in \operatorname{rel\,int} P \cap \mathcal{Q}$. This amounts to a linear programming problem with d variables and $|\mathcal{H}|$ constraints, and hence it can be solved in $O(|\mathcal{H}|)$ sequential time [12]. Note that the size of $\mathcal{H}$ is bounded by the number of oracle calls.

ii. If $\hat{L}_{\{\boldsymbol{m}\}}$ is not constant on R^d, that is, not all of $m_1, m_2, \ldots, m_d$ equal zero, then g is unbounded. Otherwise,

iii. consider $g(\boldsymbol{\lambda}^*) = m_{d+1}$. The function $\hat{L}_{\{\boldsymbol{m}\}}$ is a weak approximation of g, and $P = \Lambda$. Hence, $\boldsymbol{\lambda}^* \in \operatorname{rel\,int}\Lambda$. Output $\boldsymbol{\lambda}^*$ and $\mathcal{C} = \{\boldsymbol{m}\}$.

6 Correctness

If an oracle call results in a solution during Step 1 of Algorithm 5.2, then correctness follows by induction on the dimension. We now assume that no oracle call resulted in a solution during Step 1. In this case, a collection $\mathcal{H}$ of closed halfspaces is obtained. Recall that if an oracle call on a hyperplane H did not result in a solution, then the halfspace F returned has the following properties: (i) if the function g is bounded then $\Lambda \subset F$ but $\Lambda \not\subset H$, (ii) if the function g is unbounded, then it must be bounded on the hyperplane H, and unbounded on the halfspace F. Let P be the polyhedron $P = \bigcap_{F \in \mathcal{H}} F$. It follows that if g is bounded then $P \supset \Lambda$, and if g is unbounded then it must be bounded outside and on the boundary of P. Note that P must be of full dimension ($\dim P = d$), for if not, then it must be contained in one of the query hyperplanes, which contradicts the previous statement.

Observe that for all pairs $\boldsymbol{a}_1, \boldsymbol{a}_2$ of vectors compared by the lifted computation, one of the following must hold: either $\boldsymbol{a}_1 <_\Lambda \boldsymbol{a}_2$ and $\boldsymbol{a}_1 <_P \boldsymbol{a}_2$, or $\boldsymbol{a}_2 <_\Lambda \boldsymbol{a}_1$ and $\boldsymbol{a}_2 <_P \boldsymbol{a}_1$. The latter is obvious when we call the oracle to resolve each hyperplane query, and it is easy to see that it still holds when we employ the multi-dimensional search technique (see Problem 4.2 and Proposition 4.3) and solve these hyperplane queries by a smaller number of oracle calls. Thus, the piece $\boldsymbol{m}$ (the maximizer) found by the lifted computation must satisfy $\boldsymbol{m} <_\Lambda \boldsymbol{c}$ and hence $\boldsymbol{m} <_P \boldsymbol{c}$ for all pieces $\boldsymbol{c}$ of g. It follows that $g(\boldsymbol{\lambda}) = \boldsymbol{m}^T \hat{\boldsymbol{\lambda}}$ for all $\boldsymbol{\lambda} \in P$. Thus, g is unbounded if and only if $\hat{L}_{\{\boldsymbol{m}\}}$ is not constant, and the correctness of step ii follows. To show the correctness of step iii assume that $\hat{L}_{\{\boldsymbol{m}\}}$ is constant, and thus $g = m_{d+1}$ for all $\boldsymbol{\lambda} \in P$. Since $P \supseteq \Lambda$ we have $P = \Lambda$. It follows that $\boldsymbol{\lambda}^* \in \operatorname{rel\,int} \Lambda$, $\operatorname{aff} \Lambda = R^d$, and $\hat{L}_{\{\boldsymbol{m}\}}$ is a minimal weak approximation of g.

7 Complexity

Consider the algorithm $\mathcal{A}$. Suppose that the $C(\mathcal{A})$ comparisons performed by $\mathcal{A}$ can be divided into r phases, where C_i independent comparisons are performed during phase i ($i = 1, \ldots, r$). It follows from Proposition 4.3, that the lifted computation can be implemented in such a way that it performs $\gamma(d) \sum_{i=1}^r \lceil \log C_i \rceil$ oracle calls. It follows from Theorem 3.1 that each oracle call involves three recursive calls to instances of Problem 2.8 of lower dimension. The piecewise affine algorithms that correspond to these instances have the same number of comparisons as $\mathcal{A}$, divided into phases in the same way, and $O(d)$ times more operations. Thus, the total number of operations needed for the lifted computation is

$$d^3 \gamma(d) k T(\mathcal{A}) (\sum_{i=1}^{r} \lceil \log C_i \rceil)^d \ .$$

The number of parallel phases needed in the above computation is bounded by the product of the number of phases of the algorithm $\mathcal{A}$ with $\sum_{i=1}^r \lceil \log C_i \rceil)^d$. If the algorithm $\mathcal{A}$ is inherently sequential, then the total number of operations is $O(kT(\mathcal{A}) C(\mathcal{A})^d)$.

8 Parametric extensions of problems

The technique described in this paper was employed in [3, 7] to get algorithms for the parametric extensions of the minimum cycle and the minimum cycle-mean problems. This technique can be applied to a variety of other problems, where we consider a strongly polynomial algorithm for a problem and obtain a strongly polynomial algorithm for a parametric extension of the problem (when the number of parameters is fixed). We state the conditions where this technique is applicable and present applications.

Definition 8.1 [Parametric extensions]

i. A *problem* $S : \mathcal{P} \to R$ is a mapping from a set $\mathcal{P}$ of instances into the set of real numbers. We say that $S(P)$ is the *solution* of the problem for the instance $P \in \mathcal{P}$. Suppose that every instance $P \in \mathcal{P}$ has a *size* $\|P\|$ associated with it. The size of an instance is not necessarily defined to be the number of bits in its representation. It may be any natural parameter (for example, the number of edges in a weighted graph).

ii. Let $\mathcal{A}$ be an algorithm that computes $S(P)$. Denote by $T_{\mathcal{A}}(P)$ the number of elementary operations the algorithm performs on the instance P. The algorithm $\mathcal{A}$ is *polynomial* if $T_{\mathcal{A}}(P) = O(p(\|P\|))$ for some polynomial $p(\bullet)$.

iii. A *d-parametric extension* $P^d = (\mathcal{M}, \mathcal{Q})$ of $\mathcal{P}$ is defined as follows, where $\mathcal{Q} \subset R^d$ is a polyhedron given as an intersection of k halfspaces, and $\mathcal{M} : \mathcal{Q} \to \mathcal{P}$ is a mapping from points $\boldsymbol{\lambda} \in \mathcal{Q}$ to instances of $\mathcal{P}$. The extension P^d corresponds to a subset of instances $\{\mathcal{M}(\boldsymbol{\lambda}) \mid \boldsymbol{\lambda} \in \mathcal{Q}\} \subset \mathcal{P}$. We refer to $\mathcal{M}(\boldsymbol{\lambda}) \in \mathcal{P}$ as the instance of $\mathcal{P}$ *induced* by $\boldsymbol{\lambda}$. For an extension P^d, we define $g : \mathcal{Q} \to R$ as a mapping from vectors $\boldsymbol{\lambda} \in \mathcal{Q}$ to the solution of the corresponding induced instance $g(\boldsymbol{\lambda}) = S(\mathcal{M}(\boldsymbol{\lambda}))$. A *solution of the parametric extension* P^d is defined as follows. Consider the maximum of $g(\boldsymbol{\lambda})$. If it is finite, a solution consists of the maximum and a vector $\boldsymbol{\lambda} \in R^d$ that belongs to the relative interior of the set of vectors which maximize S. Formally, if $\mathcal{Q}$ is empty or if $S(\mathcal{M}(\boldsymbol{\lambda}))$ is unbounded on $\mathcal{Q}$, these facts are recognized. Otherwise, a pair $(m, \boldsymbol{\lambda}^*) \in R \times R^d$, where $m = \max_{\lambda \in \mathcal{Q}} g(\boldsymbol{\lambda})$, and $\boldsymbol{\lambda}^* \in \operatorname{rel\,int}\{\boldsymbol{\lambda} \mid g(\boldsymbol{\lambda}) = m\}$ is computed. We denote $T = \max_{\lambda \in \mathcal{Q}} T_{\mathcal{A}}(\mathcal{M}(\boldsymbol{\lambda}))$.

Theorem 8.2 *Let $S : \mathcal{P} \to R$ be a problem in the sense of Definition 8.1. Let $\mathcal{A}$ be an algorithm that evaluates S, and let $P^d = (\mathcal{M}, \mathcal{Q})$ (where $|\mathcal{Q}| = k$) be a corresponding parametric extension. We assume that*

i. the function g is concave,

ii. the mapping $\mathcal{M}$ is computable by a piecewise affine algorithm $\mathcal{A}_{\mathcal{M}}$ (see Definition 2.2) in less than T operations, and *iii. the combined algorithm which computes an instance $\mathcal{A}_{\mathcal{M}}(\boldsymbol{\lambda}) \in \mathcal{P}$ and applies $\mathcal{A}$ to $\mathcal{A}_{\mathcal{M}}(\boldsymbol{\lambda})$, is piecewise affine.*

Denote by C the maximum (over $\boldsymbol{\lambda} \in \mathcal{Q}$) number of comparisons performed by the combined algorithm. Suppose the comparisons can be divided into r sets of sizes $C_1, \ldots, C_r$ ($C = \sum_{i=1}^{r} C_i$) such that the algorithm runs in r phases, where C_i independent comparisons are performed in phase i.

Under these conditions, the d-parametric extension P^d can be solved using

$$\beta(d)kT(\sum_{i=1}^{r}\lceil \log C_i \rceil)^d \text{ operations, where } \beta(d) = 3^{O(d^2)} .$$

Remark 8.3 In the above formulation we defined a problem as a mapping into the set of real numbers $S : \mathcal{P} \rightarrow R$. The results generalize to cases where the range of S is R^{ℓ} for $\ell > 1$ and the notions of maximum and concavity of g are defined with respect to the lexicographic order as discussed in the introduction.

Below we present some applications of Theorem 8.2. Additional applications were found by Norton, Plotkin, and Tardos [13].

Adding variables to LP's with two variables per inequality Linear programming problems with at most two variables in each constraint and in the objective function were shown to have a strongly polynomial time algorithm by Megiddo [11]. Lueker, Megiddo and Ramachandran [10] gave a polylogarithmic time parallel algorithm for the problem which uses a quasipolynomial number of processors. The best known time bounds for the problem were given in [6, 2]. Cosares, using nested parametrization, extended Megiddo's strong polynomiality result to allow objective functions which have a fixed number of nonzero coefficients. This result can be further extended to include the following. For a fixed d, we consider linear programming problems as above, but we allow certain d additional variables to appear anywhere in the constraints and in the objective function without being "counted." This problem is a d-parameter extension of the two variables per constraint problem, where the "parameters" are the d additional variables. For each choice of values for the parameters we have a corresponding induced system with two variable per constraint. It is easy to verify that the conditions of Theorem 8.2 hold. Hence, this class of problems also has a strongly polynomial time algorithm, and a polylogarithmic time parallel algorithm which uses a quasipolynomial number of processors.

Parametric flow problems Theorem 8.2 was applied in [5, 8] to generate strongly polynomial algorithms for parametric flow problems with a fixed number of parameters and to some constrained flow problems with a fixed number of additional constraints. Complementing results showing the P-completeness of these problems when the number of parameters is not fixed, were also given.

References

[1] K. L. Clarkson. Linear programming in $O(n \times 3^{d^2})$ time. *Information Processing Let.*, 22:21–27, 1986.

[2] E. Cohen. *Combinatorial Algorithms for Optimization Problems*. PhD thesis, Department of Computer Science, Stanford University, Stanford, Ca., 1991.

[3] E. Cohen and N. Megiddo. Strongly polynomial and NC algorithms for detecting cycles in dynamic graphs. In *Proc. 21st Annual ACM Symposium on Theory of Computing*, pages 523–534. ACM, 1989.

[4] E. Cohen and N. Megiddo. Maximizing concave functions in fixed dimension. Technical Report RJ 7656 (71103), IBM Almaden Research Center, San Jose, CA 95120-6099, August 1990.

[5] E. Cohen and N. Megiddo. Complexity analysis and algorithms for some flow problems. In *Proc. 2nd ACM-SIAM Symposium on Discrete Algorithms*, pages 120–130. ACM-SIAM, 1991.

[6] E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. In *Proc. 23rd Annual ACM Symposium on Theory of Computing*, pages 145–155. ACM, 1991.

[7] E. Cohen and N. Megiddo. Strongly polynomial time and NC algorithms for detecting cycles in periodic graphs. *J. Assoc. Comput. Mach.*, 1992. To appear.

[8] E. Cohen and N. Megiddo. Complexity analysis and algorithms for some flow problems. *Algorithmica*, 1993. To appear.

[9] M. E. Dyer. On a multidimensional search technique and its application to the Euclidean one-center problem. *SIAM J. Comput.*, 15:725–738, 1986.

[10] G. S. Lueker, N. Megiddo, and V. Ramachandran. Linear programming with two variables per inequality in poly log time. *SIAM J. Comput.*, 19(6):1000–1010, 1990.

[11] N. Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM J. Comput.*, 12:347–353, 1983.

[12] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. Assoc. Comput. Mach.*, 31:114–127, 1984.

[13] C. H. Norton, S. A. Plotkin, and É. Tardos. Using separation algorithms in fixed dimension. In *Proc. 1st ACM-SIAM Symposium on Discrete Algorithms*, pages 377–387. ACM-SIAM, 1990.