APPROXIMATION ALGORITHMS FOR DIFFERENCE OF CONVEX (DC) PROGRAMMING PROBLEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

By
Fahaar Mansoor Pirani
July 2023

Approximation Algorithms for Difference of Convex (DC) Programming Problems
By Fahaar Mansoor Pirani
July 2023

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Firdevs Ulus(Advisor)

Taghi Khaniyev

Hamdullah Yücel

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

APPROXIMATION ALGORITHMS FOR DIFFERENCE OF CONVEX (DC) PROGRAMMING PROBLEMS

Fahaar Mansoor Pirani
M.S. in Industrial Engineering
Advisor: Firdevs Ulus
July 2023

This thesis is concerned with Difference of Convex (DC) programming problems and approximation algorithms to solve them. There is an existing exact algorithm that solves DC programming problems if one component of the DC function is polyhedral convex [1]. Motivated by this, first, we propose an algorithm (Algorithm 1) for generating an ϵ-polyhedral underestimator of a convex function g. The algorithm starts with a polyhedral underestimator of g and the epigraph of the current underestimator is intersected with a single halfspace in each iteration to obtain a better approximation. We prove the correctness and establish the convergence rate of Algorithm 1. We also propose a modified variant (Algorithm 2) in which multiple halfspaces are used to update the epigraph of current approximation in each iteration. In addition to its correctness, we prove that Algorithm 2 terminates after finitely many iterations. We show that after obtaining an ϵ-polyhedral underestimator of the first component of a DC function, the algorithm from [1] can be applied to compute an ϵ-solution of the DC programming problem.

We also propose an algorithm (Algorithm 3) for solving DC programming problems directly. In each iteration, Algorithm 3 updates the polyhedral underestimator of g locally while searching for an ϵ-solution to the DC problem directly. We prove that the algorithm stops after finitely many iterations and it returns an ϵ-solution to the DC programming problem. Moreover, the sequence $\{x^k\}_{k\geq 0}$ outputted by Algorithm 3 converges to a global minimizer of the DC problem when ϵ is set to zero. The computational results, obtained using some test examples from [2], show comparable performance of Algorithms 1, 2 and 3 with respect to two DC programming algorithms from the literature.

Keywords: DC Programming Problems, Polyhedral Approximation, Algorithms.

ÖZET

DIŞBÜKEY FARKI (DC) PROGRAMLAMA PROBLEMLERİ İÇİN YAKLAŞIKLAMA ALGORİTMALARI

Fahaar Mansoor Pirani
Endüstri Mühendisliği, Yüksek Lisans
Tez Danışmanı: Firdevs Ulus
Temmuz 2023

Bu tezde dışbükey farkı (DC) programlama problemleri ve yaklaşıklama algoritmaları çalışılmıştır. DC fonksiyonunun bir bileşeni dışbükey çokyüzlü olduğunda DC programlama problemlerini çözen mevcut bir kesin algoritma bulunmaktadır [1]. Bundan yola çıkarak bu tezde, öncelikle bir dışbükey g fonksiyonunun ϵ-çokyüzlü azımsayıcısını oluşturmak için bir dış yaklaşıklama algoritması (Algoritma 1) önerilmiştir. Algoritma 1, g'nin bir çokyüzlü azımsayıcısı ile başlar ve her tekrarda, daha iyi bir yaklaşık elde etmek için mevcut yaklaşık çokyüzlü fonksiyonun epigrafını tek bir yarıuzay ile keser. Algoritma 1'in doğruluğu kanıtlanmış ve yaklaşıklama oranı hesaplanmıştır. Buna ek olarak Algoritma 1'in değiştirilmiş bir varyantı (Algoritma 2) önerilmiştir. Algoritma 2'nin temel farkı her tekrarda mevcut çokyüzlü fonksiyonu güncellerken bir değil birden fazla yarıuzayın tek seferde epigraf ile kesiştirilmesidir. Doğruluğuna ek olarak, Algoritma 2'nin sonlu tekrardan sonra sona erdiği kanıtlanmıştır. DC fonksiyonunun ilk bileşeninin ϵ-çokyüzlü azımsayıcısı elde edildikten sonra, [1]'deki algoritmanın DC programlama problemine bir ϵ-çözüm bulmak için uygulanabildiği gösterilmiştir.

Tezde ayrıca DC programlarını doğrudan çözmek için üçüncü bir algoritma (Algoritma 3) önerilmiştir. Algoritma 3, diğer iki algoritma gibi g'ye çokyüzlü azımsayıcı oluşturarak ilerler. Ancak bu algoritma her tekrarda doğrudan DC programlama probleminin bir ϵ-çözümünü arar ve bunu yaparken g'nin çokyüzlü azımsayıcısını yerel olarak günceller. Algoritmanın sonlu sayıda yinelemeden sonra durduğunu ve DC programlama problemine bir ϵ-çözüm bulduğu kanıtlanmıştır. Buna ek olarak, ϵ sıfıra eşitlendiğinde, Algoritma 3'ün çıktısı olan $\{x^k\}_{k\geq 0}$ dizisinin DC probleminin evrensel azımsayıcısına yakınsadığı kanıtlanmıştır. Bazı test problemleri kullanılarak elde edilen deneysel sonuçlar, bu tezde önerilen algoritmaların mevcut iki DC programlama algoritmasına göre karşılaştırılabilir performansa sahip olduğunu göstermektedir.

Anahtar sözcükler: DC Programlama Problemleri, Çokyüzlü Yaklaşıklama, Algoritmalar.

Acknowledgement

Foremost, I would like to express my deepest gratitude to my advisor Asst. Prof. Firdevs Ulus for her continuous guidance and never ending patience and support. I feel extremely lucky that she took me under her supervision since my undergraduate study. She taught me the importance of following and unravelling my ideas to become a good researcher. I am extremely grateful that she trusted me and guided me through the subject from the very basics till the completion of this thesis. This thesis would not have been possible without her endless guidance and support. I feel proud of working under her supervision.

I am deeply indebted to department chairs Prof. Mehmet Selim Aktürk and Prof. Bahar Yetiş Kara for their kind and continuous support which always remained available through their prompt and favorable responses. I am highly grateful to my thesis committee members Asst. Prof Taghi Khaniyev and Assoc. Prof Hamdullah Yücel for taking time out of their busy schedules to review my thesis and for their valuable suggestions and insights which helped to improve and refine my work. I am also grateful to the faculty members of the Bilkent IE department throughout my undergraduate and graduate studies, especially Asst. Prof. Çağın Ararat. All I know about probability theory and stochastics is due to his insightful lectures both in undergradaute and master's, which in a way prompted me to continue pursuing a career in academia. I am also grateful to Prof. Yosum Kurtulmaz, from the Department of Mathematics Bilkent University, for taking out time to attend my thesis presentation.

My words cannot express my gratitude to my best friend and fellow master's student Wissam Al-Ali who was there in all my good and bad times since my undergraduate study. I am extremely grateful for his companionship throughout these six years and for his valuable assistance on issues that I faced throughout this journey. I would also like to extend my sincere thanks to all my friends (Turkish, Pakistani and international) for making my undergraduate and master's journey a remarkable one and for all their assistance throughout this journey of six years. I would like to list a few names; Wissam Al-Ali, Kürşad Ali Akdoğan, Yunus Emre Yürdagül, İrem Ünal, Özlen Ataç, Kağan Çalıkoğlu, Erdil Şen, Ferman Dağ, Hakan Dorukhan Yeşilli, Çerağ Oğuztüzün, Bora Çetin, Ayşe Beyza Çanakçı, Sena Aydın, Hilal Khan, Asfandyar Khan, Muzaffar Rafique, Abdullah Bin Aamir, Mustafa Zaki, Usman

Naeem, Talha Ejaz, Waqar Ahmad, Najeeb Ullah, Badar Munir, Ahsan Tauseef, Umar Rauf, Saad Rao, Osama Zafar, Hamza Islam and many others. In addition, I would like to thank my Operations Research (OR) club family for making me feel at home. I would also like to list their names as well; Arda Şahinoğlu, Wasiq Ahmed, Efe Umut Ateş, Deniz Aralan, Arca Şahin, Nilsu Sözer, Aslı Koper and others. Moreover, I would also like to express my gratitude to Muhammad Umer, a PhD graduate from Bilkent IE department, whose valuable suggestions and contributions also helped me to complete this thesis.

Most importantly, I would like to extend my gratitude to my parents and my family. I am what I am today because of my parents. Whenever I got stuck in anything, my mother always listened to my pleas, calmed me down, and provided me with a solution. My father played a huge role throughout my academic journey. He taught me the importance of time and being self-reliant. They have invested their lives in me and I would never be able to pay them back for what they have sacrificed for me. I am also grateful to my brother Ali Mohammad and my late grandmother Laila Salim for their continuous and unwavering support. Lastly, I would like to extend my gratitude to my relatives in U.S. who, although miles apart, have always helped me in any way possible throughout this vast journey of six years. This work would not have been possible without my family's continuous support, encouragement and their never-ending trust in me.

To my parents Farida and Mansoor,
In memory of my grandmother Laila; may her soul rest in peace.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we consider the literature review concerning DC programming problems, and various existing approaches that have been used to solve such problems. Then, we discuss the contributions of this thesis.

1.1 Literature Review and Motivation

In non-convex optimization, the decision maker considers a non-convex objective function. The absence of convexity in these problems creates different kinds of complications and complexities. For example, it is hard to obtain global solutions and to distinguish between the local and global solutions. Consequently, the computational complexity, when passing from convex to non-convex programming increases rapidly. Over the last thirty years, a variety of non-convex optimization algorithms and methods have been developed. These can be classified into two groups: global and local. Global approaches, such as cutting planes, branch and bound, etc., guarantee finding a global optimal solution or at least global ϵ-solution [5]. However, these are quite expensive and time-consuming, especially in a large-scale setting. Local approaches, on the other hand, are relatively faster. However, they may not return global solutions [6].

DC programming is a class of optimization problems that falls under the category of non-convex and global optimization. A function $f : \mathbb{R}^n \to \mathbb{R}$ is a DC function on a convex set

$X \subseteq \mathbb{R}^n$, if it can be written as a difference of two continuous convex functions $g : \mathbb{R}^n \to \mathbb{R}, h : \mathbb{R}^n \to \mathbb{R}$,

$$f(x) = g(x) - h(x), \quad x \in X.$$

We say that $g - h$ is a DC decomposition of f. In general, the formulation for a DC program is as follows:

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leq 0, \quad i = 1, \ldots, m, \\ & f_{m+j}(x) \leq 0, \quad j = 1, \ldots, r, \end{aligned}$$

where $f_0, f_i, i = 1, \ldots, m$ are DC functions on $\mathbb{R}^n$ and $f_{m+j}, j = 1, \ldots, r$ are convex functions on $\mathbb{R}^n$. In this thesis, we consider the following DC programming problem

$$\min_{x \in X}(g(x) - h(x)), \tag{P}$$

where g and h are convex functions on X. For theoretical results, X is a convex compact subset of $\mathbb{R}^n$. For computational results, X will be a box in $\mathbb{R}^n$. Clearly, any convex optimization problem is a DC programming problem. However, methods related to convex programs do not work since the DC objective function is not convex in general.

Many large-scale and real-life optimization problems can be transformed into equivalent DC programming problems. Some examples can be seen in production–transportation planning, location planning, engineering design, cluster analysis, multi-level programming and multi-objective programming [2]. DC programming has been also very useful in solving non-convex problems from many fields of applied sciences including data science, communication systems, biology, finance, logistics, supply chain management, etc., and remarkable results have been achieved. DC algorithms have been successful in solving highly structured problems, see Holmberg and Tuy [7], Chen, Hansen, Jaumard and Tuy [8], Horst and Thoai [9].

Non-convex optimization and global optimization have long posed significant challenges due to the complex nature of the objective and constraint functions involved. In 1985, Pham Dinh Tao [10] introduced DC Programming which have become an important and powerful tool of non-convex optimization, see [5, 11, 12, 13, 14]. This development was an extension of his previous works on convex maximization since 1974 [10]. Indeed, the initial groundwork

for convex maximization was laid by Hoang Tuy [15] in 1964. Tuy studied maximization of convex functions over a polyhedral convex set and proposed solution algorithms. Tao also introduced the algorithmic counterpart of DC programming, DCA (DC Algorithm), which have become an important tool towards solving different DC programming problems to find local optimal solutions, since 1993 as a result of intensive research on this field, see [16, 17, 18, 19, 20].

The popularity of DC programming and DCA can be attributed to several key factors mentioned in [10]. DCA's strength lies in its ability to break down non-convex functions into smaller convex subproblems, which gives way to a wide range of possibilities for problem formulations and solution techniques. The implementation is relatively easy and simple. Additionally, DCA's flexibility allows for adaptation to specific problem structures, making it a suitable choice for addressing various optimization problems. By decomposing non-convex functions into convex subproblems, DCA can take advantage of efficient convex optimization techniques, leading to faster convergence and reduced computational complexity. This makes DC programming and DCA applicable to real-world applications that involve complex and high-dimensional problems.

Following Tao's introduction and implementation of DCA [10], different proposed methods to solve DC programming problems have been developed. Horst and Thoai [9] and Hoai An and Tao [5] mentioned two approaches of solving a DC programming problem by a *Combinatorial* approach and a *Convex Analysis* approach. The Combinatorial approach is quite old and uses tools from combinatorial optimization. The common tool is the Branch and Bound technique which is widely used in discrete and combinatorial optimization. In the recent decade, progress has been made tremendously in terms of computations. One can globally solve large-scale low rank non-convex programs, see Konno, Thach, and Tuy [21]. However, these algorithms do not solve real-life DC programs. On the other hand, the Convex Analysis approach makes use of the outer approximation algorithms. The idea is based on an iterative approximation of a convex feasible set or domain by a sequence of polyhedral convex sets enclosing it [9]. The approach seems to have risen in the works of Pham Dinh Tao [22] on the computation of bound-norms of matrices. The main idea is to use the duality theory for DC functions, which was first introduced by Toland in 1979 [23]. Over the last thirty years, the convex analysis approach has gained recognition with the discovery of different methods. Konno, Thach, and Tuy [21], Horst and Tuy [24], Strekalovsky and Tsevendorj [25], An and Tao [26] introduced algorithms for solving DC

programming problems using the convex analysis approach.

Note that DCA and many alternative algorithms from the literature guarantee finding a local minimum of the DC programming problem. Indeed, the problem of finding a global minimum of DC programming problems is NP-hard [2]. This makes the exact solution approaches for DC programming computationally expensive, even for small sized problems. Therefore, the speed and efficiency of the algorithms is vital. Hence, a suitable combination between branch and bound algorithms and outer approximation algorithms can lead to a more productive and efficient result. This hybrid approach has been used to solve different kinds of convex and non-convex programs, see [27, 28, 29].

Polyhedral DC program is a DC program in which one of the component functions g or h is polyhedral convex. Polyhedral DC programming has played a central role in the realm of non-convex and global optimization. It has interesting properties on local optimality conditions and the finite convergence of DCA [6]. In 2017, Löhne and Wagner [1] proposed a method for finding an optimal solution to polyhedral DC programming problems by solving a polyhedral projection problem and finitely many convex programs. Moreover, in 2020, Löhne and Dahl proposed solving polyhedral DC programming problems using concave minimization techniques [30]. The polyhedral DC program is transformed into a concave minimization problem under linear constraints. Although, many authors consider a compact feasible set to guarantee an optimal solution in concave minimization problems, Löhne and Dahl solved the problem using a non-compact feasible set.

1.2 Algorithms for Generating Polyhedral Underestimators of Convex Functions

Motivated by [1], in this thesis, we first propose algorithms to generate polyhedral underestimators to convex functions. The idea is to transform the DC programming problem into an approximate polyhedral DC programming problem and use the methods from [1] to solve the DC programming problem approximately. For this purpose, we propose algorithms for generating a sequence of polyhedral underestimators to g. Indeed, these underestimators consist of supporting halfspaces, which act as support functions. Beliakov [31] also suggested various other choices of support functions that can be used to create different underestimators,

based on the function's properties on which it is being applied. In this thesis, we propose two algorithms that use affine minorants, as the support functions, to approximate g. This results in the underestimator being a polyhedral convex function. Algorithm 1 starts with an initial polyhedral underestimator of g and in each iteration computes the vertices of the epigraph of the current underestimator. Using the vertex that is farthest away from the epigraph of g, it updates the underestimator until the vertical distance between the functions is sufficiently small.

We prove that Algorithm 1 is correct and we also estimate the convergence rate of it. The motivation for the convergence rate comes from Ararat, Ulus, and Umer [32], who use the work of Lotov et al [33] and Kamenev [34] to prove the convergence rate of one of their algorithms to solve convex vector optimization problems. The idea behind the proof is that a suitable compact subset of the epigraph of the original function is approximated by a particular sequence of polytopes. In particular, after obtaining a compact subset of the epigraph, and satisfying certain conditions, we use a result available in the literature [33, Chapter 8] and [34].

Algorithm 2 is similar to Algorithm 1. Different from it, all vertices of the epigraph of the current approximation is considered to update the underestimator. We also prove the correctness and finiteness of Algorithm 2 for a given $\epsilon > 0$. The idea of the finiteness follows a similar approach by Ararat, Ulus, and Umer [35], who prove the finiteness of an algorithm for solving convex vector optimization problems. We introduce a hypervolume argument which explores the relationship between certain supporting halfspaces of the epigraph and certain norm balls, see Lemma 4.2.2. In doing so, we prove that throughout the iterations of Algorithm 2, one can construct non-overlapping subsets of the compact feasible set with fixed volume.

Both Algorithms 1 and 2 provide a polyhedral underestimator to a given convex function g. However, based on the computational tests given in Chapter 6, Algorithm 2 takes much less computational time to construct a polyhedral underestimator than Algorithm 1. This is because, at each iteration, Algorithm 2 intersects multiple supporting halfspaces, instead of one, to the current approximation. In doing so, the polyhedral approximations outputted by Algorithm 2 are much stronger than the ones outputted by Algorithm 1.

1.3 A DC Algorithm based on Polyhedral Underestimators

Next, we propose an algorithm, Algorithm 3, to solve DC programming problems by constructing polyhedral underestimators to g. While Algorithms 1 and 2 are naive approaches for generating polyhedral underestimators, Algorithm 3 uses the hybrid approach. At each iteration, it generates polyhedral underestimators to g, followed by solving a polyhedral DC programming problem, using methods from [1], see Appendix A.1 for the method by Löhne and Wagner. While Algorithms 1 and 2 ensure that the resulting polyhedral underestimator is within the vertical distance $\epsilon > 0$, Algorithm 3 does not constructs an ϵ-polyhedral underestimator. Instead, it traverses locally and updates the polyhedral underestimator of g, while searching for an ϵ-solution of the DC programming problem. We also prove the correctness and convergence of Algorithm 3. While the original DCA, introduced by Pham Dinh Tao, finds a local minimizer to the DC programming problem [10, Section 2], Algorithm 3 finds a global ϵ-minimizer. We show that for a sequence of points in the compact feasible set X, $\{x^k\}_{k\geq 0}$, outputted by Algorithm 3, the limit point of this sequence is a global minimizer of the original DC programming problem that lies in X, see Theorem 5.2.2. In addition, we also prove that for $\epsilon > 0$, Algorithm 3 terminates after finitely many iterations.

1.4 Thesis Outline

The remainder of the thesis is as follows. In Section 2.1, we introduce notations and recall some concepts from real and convex analysis and other relevant areas. In Section 2.2, we introduce the problem and provide the proof of a lemma which will be useful in proving other results of our algorithms in the subsequent chapters. Chapter 3 is devoted to Algorithms 1 (Section 3.1) and 2 (Section 3.2), respectively. We also prove that they work correctly. This is followed by a detailed convergence analysis of these algorithms in Chapter 4. This includes convergence analysis of Algorithm 1 in Section 4.1 followed by the finiteness result of Algorithm 2 in Section 4.2, respectively. Then, in Chapter 5, we provide a detailed explanation of Algorithm 3 which solves DC programming problems. We also show its correctness, finiteness, and provide convergence results. Finally, we discuss the computational performance of Algorithms 1, 2 and 3 on several examples in Chapter 6 and future research directions

in Chapter 7. Appendix A recalls some important methods and algorithms from the literature. This includes the method by Löhne and Wagner [1] in Appendix A.1 followed by the Pijavski-Shubert [3, 4] and DECEAM [2] methods in Sections A.2.1 and A.2.2, respectively. In Appendix B, we provide the codes for Algorithms 1, 2, and 3 written in MATLAB.

Chapter 2

Problem Definition and Formulation

In this chapter, we introduce some definitions and notations and describe the Difference of Convex (DC) Programming Problem. Moreover, we present some results which will be used to prove the main theorems of the thesis.

2.1 Preliminaries

We denote by $\mathbb{R}^n$, the n-dimensional Euclidean space. The standard basis of $\mathbb{R}^n$ is formed by the set of n unit vectors $\{e^1, e^2, \ldots, e^n\}$ whose components are all zeros, except one. In particular, $e^i \in \mathbb{R}^n$ is a vector in $\mathbb{R}^n$ such that $e^i_i = 1$ and $e^i_j = 0$ for all $j \neq i$.

Let $S, T \subseteq \mathbb{R}^n$ be nonempty sets and $\mu \in \mathbb{R}$. The Minkowski operations of the sets is defined $S + T \coloneqq \{x_1 + x_2 \mid x_1 \in S, x_2 \in T\}, \mu S \coloneqq \{\mu x \mid x \in S\}, S - T \coloneqq S + (-1)T$. For a set $S \subseteq \mathbb{R}^n$, its convex hull, conic hull, interior, relative interior, boundary are denoted by $\operatorname{conv} S, \operatorname{cone} S, \operatorname{int} S, \operatorname{ri} S, \operatorname{bd} S$, respectively. A recession direction of S is a vector $k \in \mathbb{R}^n \setminus \{0\}$ satisfying $S + \operatorname{cone}\{k\} \subseteq S$. The *recession cone* of S is the set of all recession directions of S, $\operatorname{recc} S = \{k \in \mathbb{R}^n \mid \forall s \in S, \forall \mu \geq 0 : s + \mu k \in S\}$.

Suppose that S is a convex set and let $x \in S, w \in \mathbb{R}^n \setminus \{0\}$. If $w^\mathsf{T} x = \inf_{y \in S} w^\mathsf{T} y$, then the set $\{y \in \mathbb{R}^n \mid w^\mathsf{T} y = w^\mathsf{T} x\}$ is called a *supporting hyperplane* of S at x, while the set $\{y \in \mathbb{R}^n \mid w^\mathsf{T} y \geq w^\mathsf{T} x\} \supseteq S$ is called a *supporting halfspace* of S at x.

Suppose S is a nonempty closed *polyhedral convex set*. Then S can be represented as the intersection of finite number of halfspaces, that is, as $S = \bigcap_{i=1}^{r} \{y \in \mathbb{R}^n \mid (w^i)^\mathsf{T} y \geq a^i\}$ for some $r \in \mathbb{N}, w^i \in \mathbb{R}^n \setminus \{0\}$ and $a^i \in \mathbb{R}$. This is known as an *H-representation* of S. Alternately, S can also be represented by a finite set of vertices $\{y^1, \ldots, y^s\} \subseteq \mathbb{R}^n$ and a finite set of directions $\{d^1, \ldots, d^t\} \subseteq \mathbb{R}^n$ via $S = \operatorname{conv}\{y^1, \ldots, y^s\} + \operatorname{conv}\operatorname{cone}\{d^1, \ldots, d^t\}$, which is known as a *V-representation* of S. Throughout the thesis, vertices of S is denoted by $\operatorname{vert} S \subseteq \mathbb{R}^n$.

Let $\bar{\mathbb{R}}$ denote the extended real line, that is, $\bar{\mathbb{R}} \coloneqq \mathbb{R} \cup \{\pm\infty\}$ and $f : \mathbb{R}^n \to \bar{\mathbb{R}}$ be a function. The *effective domain* of f is $\operatorname{dom} f \coloneqq \{x \in \mathbb{R}^n \mid f(x) < \infty\}$. The function f is said to be *proper* if

(a) $f(x) > -\infty$ for every $x \in \operatorname{dom} f$, and

(b) there exists some point $x_0 \in \operatorname{dom} f$ such that $f(x_0) \in \mathbb{R}$.

The *epigraph* of f is defined as $\operatorname{epi} f \coloneqq \{(x, r) \in \mathbb{R}^n \times \mathbb{R} \mid f(x) \leq r\}$, and f is said to be a closed function if $\operatorname{epi} f$ is closed. The set $\partial f(x_0) \coloneqq \{c \in \mathbb{R}^n \mid \forall x \in \mathbb{R}^n \colon f(x) \geq f(x_0) + c^\mathsf{T}(x - x_0)\}$ is called the *subdifferential* of f at x_0. An arbitrary element of $\partial f(x_0)$ is called a *subgradient* of f at x_0 and denoted by $c(x_0)$, throughout. A proper convex function is closed if and only if it is lower semi-continuous. Additionally, a closed proper convex function f is the pointwise supremum of the collection of all affine functions h such that $h \leq f$ (called the affine minorants of f).

The *conjugate function of* f is a convex function $f^* : \mathbb{R}^n \to \mathbb{R}$ given by

$$f^*(y) := \sup_{x \in \operatorname{dom} f} \{y^\mathsf{T} x - f(x)\}$$

[36, Section 3.3]. Note that $y^\mathsf{T} x \leq f(x) + f^*(y)$ holds for all $x, y \in \mathbb{R}^n$. This is known as Fenchel's inequality. On $\mathbb{R}^n$, let $\|\cdot\|$ be an arbitrary norm, and $\|\cdot\|_*$ be its dual norm. For any $x, y \in \mathbb{R}^n$, we have $|x^\mathsf{T} y| \leq \|x\| \, \|y\|_*$. This is known as the Hölder's inequality. The conjugate function of the norm function is the indicator function of the unit ball with respect

to the dual norm [36, Example 3.26], that is,

$$\|\cdot\|^*(y) = \begin{cases} 0, & \text{if } \|y\|_* \leq 1, \\ +\infty, & \text{if } \|y\|_* > 1. \end{cases}$$

For $x \in \mathbb{R}^n$, $\epsilon > 0$, the closed ball centered at x having radius ϵ is denoted by $\mathbb{B}[x, \epsilon] := \{z \in \mathbb{R}^n \mid \|z - x\| \leq \epsilon\}$. For every $y \in \mathbb{R}^n$, the distance from a point y to a set $S \subseteq \mathbb{R}^n$ is defined as $d(y, S) := \inf_{x \in S} \|y - x\|$. Let $T \subseteq \mathbb{R}^n$. The Hausdorff distance between S and T, denoted as $\delta^H(S, T)$, is given by [37, Theorem 7.3.1]

$$\delta^H(S, T) = \max\{\sup_{x \in S} d(x, T), \sup_{y \in T} d(y, S)\}.$$

The following proposition, corollary, and the two lemmata are simple observations that will be useful throughout the thesis.

Proposition 2.1.1. *[1, Proposition 7] Let $f : \mathbb{R}^n \to \bar{\mathbb{R}}$ be a proper convex function and let $P \subseteq \mathbb{R}^n$ be a polyhedron, which has at least one vertex. Consider the following problem:*

$$\text{maximize } f(x), \quad \text{subject to} \quad x \in P. \tag{1}$$

If problem (1) has an optimal solution, then some vertex $x^* \in P$ is an optimal solution of (1).

Corollary 2.1.2. *Let $g : X \to \mathbb{R}$, $h : X \to \mathbb{R}$ be convex functions where g is also polyhedral convex. Then the following polyhedral DC problem has a solution at one of the vertices of* epi g

$$\min_{x \in X}(g(x) - h(x))$$

Lemma 2.1.3. *Let $S \subseteq \mathbb{R}^n$ be a closed nonempty set. The function $d(\cdot, S) : \mathbb{R}^n \to \mathbb{R}$ given by $d(x, S) = \inf_{s \in S} \|x - s\|$ is convex if and only if S is a convex set.*

Proof. Let S be a convex set. The function $\|x - s\|$ is convex in (x, s). Since the function is convex and S is a non-empty convex set, the function $d(x, S) = \inf_{s \in S} \|x - s\|$ is convex in x by [36, Section 3.2]. Now, assume $d(x, S)$ is a convex function. Let $x_1, x_2 \in S$ and $\lambda \in [0, 1]$. We have $d(\lambda x_1 + (1 - \lambda)x_2, S) \leq \lambda d(x_1, S) + (1 - \lambda)d(x_2, S) = 0$, since $x_1, x_2 \in S$. This implies $\lambda x_1 + (1 - \lambda)x_2 \in S$. □

A function $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is *polyhedral convex* if epi f is a polyhedral convex set.

Lemma 2.1.4. *[38, Lemma 2.2] Let S and T be convex sets in $\mathbb{R}^n$ with* $\operatorname{recc} S = \operatorname{recc} T$ *and $T \subseteq S$. If S is polyhedral convex with at least one vertex, then $\delta^H(S,T) = \max_{v \in \operatorname{vert} S} d(v,T)$, where* $\operatorname{vert} S$ *is the set of all vertices of S.*

Proof. For any two sets S, T, the Hausdorff distance is given as the following:

$$\delta^H(S,T) = \max\{\sup_{y \in S} d(y,T), \sup_{y \in T} d(y,S)\} = \sup_{y \in S} d(y,T),$$

where the last equality follows because $T \subseteq S$. Since S is a polyhedron and $d(y,T)$ is a convex function in y, $\sup_{y \in S} d(y,T)$ is a convex problem. By Proposition 2.1.1, it has an optimal solution at one of the vertices of S. Hence, $\sup_{y \in S} d(y,T) = \max_{v \in \operatorname{vert} S} d(v,T)$. □

2.2 Difference of Convex Programming

The problem that we are interested in is to obtain the global minimum of a DC function, over a convex compact set $X \subseteq \mathbb{R}^n$, that is, solving the problem

$$\min_{x \in X} (g(x) - h(x)), \tag{P}$$

where $g : \mathbb{R}^n \to \mathbb{R}$ and $h : \mathbb{R}^n \to \mathbb{R}$ are convex functions.

Assumption 2.2.1. *We assume that $X = \{x \in \mathbb{R}^n \mid \ell \leq x \leq u\}$ for some $\ell, u \in \mathbb{R}^n$ such that $\ell_i < u_i$ for all $i \in \{1, \ldots, n\}$.*

The existence of a solution to problem (P) is known under Assumption 2.2.1. However, most of the existing solution approaches guarantee finding either only local optimal solutions or near-optimal solutions in the sense of the following definition.

Definition 2.2.2. *Let $\epsilon > 0$. $\bar{x} \in X$ is an ϵ-solution to problem (P) if $g(\bar{x}) - h(\bar{x}) \leq \min_{x \in X}(g(x) - h(x)) + \epsilon$.*

The following lemma and remark will be useful for the design of the proposed algorithms, see Chapters 3 and 5.

Lemma 2.2.3. *Let $g : \mathbb{R}^n \to \mathbb{R}$ be convex and $x^* \in \mathbb{R}^n$. Then,*

$$H(g, x^*) := \{(x, t) \in \mathbb{R}^n \times \mathbb{R} \mid t - c(x^*)^\mathsf{T} x \geq g(x^*) - c(x^*)^\mathsf{T} x^*\}$$

is a supporting halfspace to $\operatorname{epi} g$ *at $(x^*, g(x^*))$, where $c(x^*) \in \partial g(x^*)$ is a subgradient of g at x^*.*

Proof. First, note that $(x^*, g(x^*)) \in \operatorname{bd} H(g, x^*)$ by construction. Moreover, for any $(x, t) \in \operatorname{epi} g$, we have

$$t \geq g(x) \geq g(x^*) + c(x^*)^\mathsf{T}(x - x^*),$$

where the second inequality is by the definition of subgradient of g at x^*. Rearranging the terms, we obtain that $(x, t) \in H(g, x^*)$. Hence, $\operatorname{epi} g \subseteq H(g, x^*)$. □

Remark 2.2.4. *Let $H(g, x^*)$ be as given in Lemma 2.2.3 for a convex function g and $x^* \in \mathbb{R}^n$. Let $s : \mathbb{R}^n \to \mathbb{R}$ be a linear function given by $s(x) := g(x^*) + c(x^*)^\mathsf{T}(x - x^*)$. Then, $H(g, x^*) = \operatorname{epi} s$ and $s(x) \leq g(x)$ for all $x \in \mathbb{R}^n$.*

Chapter 3

Algorithms for approximating a convex function

As mentioned in Chapter 1, Löhne and Wagner [1] proposed an exact algorithm to solve problem (P) if at least one of g or h is a polyhedral convex function. The main idea of the first solution approach that we propose is to find a polyhedral approximation $\bar{g}$ of g over X and to use the algorithm from [1] for finding an exact solution of problem

$$\min_{x \in X}(\bar{g}(x) - h(x)). \tag{$P_{\bar{g}}$}$$

Similarly, it is possible to obtain a polyhedral approximation $\bar{h}$ of h and to solve

$$\min_{x \in X}(g(x) - \bar{h}(x)). \tag{$P_{\bar{h}}$}$$

To start with, we define a polyhedral approximation of a convex function with required properties as follows.

Definition 3.0.1. *Let $\epsilon > 0$ and $g : \mathbb{R}^n \to \mathbb{R}$ be a convex function on a convex set $X \subseteq \mathbb{R}^n$. A polyhedral convex function $\bar{g} : \mathbb{R}^n \to \mathbb{R}$ is called an* ϵ-polyhedral underestimator of g on X *if for all $x \in X$, it satisfies*

$$0 \leq g(x) - \bar{g}(x) \leq \epsilon.$$

In Theorem 3.0.3 below, we show that if $\bar{g}$ (resp. $\bar{h}$) is an ϵ-polyhedral underestimator of

g (resp. h) on X, then solving $(P_{\bar{g}})$ (resp. $(P_{\bar{h}})$) yields an ϵ-solution to problem (P). The following lemma will be useful for other results of this thesis.

Lemma 3.0.2. *Let $f_1 : \mathbb{R}^n \to \mathbb{R}$ and $f_2 : \mathbb{R}^n \to \mathbb{R}$ be continuous functions on $\mathbb{R}^n$ and $X \subseteq \mathbb{R}^n$. Then the following holds*

$$\sup_{x\in X} f_2(x) - \sup_{x\in X} f_1(x) \leq \sup_{x\in X}\{f_2(x) - f_1(x)\}.$$

Proof. Observe that

$$\sup_{x\in X} f_2(x) = \sup_{x\in X}\{(f_2(x) - f_1(x)) + f_1(x)\} \leq \sup_{x\in X}\{f_2(x) - f_1(x)\} + \sup_{x\in X} f_1(x),$$

holds by triangle inequality. Rearranging the terms gives us the required inequality. □

Theorem 3.0.3. *Let $\epsilon > 0$, $g : \mathbb{R}^n \to \mathbb{R}$ and $h : \mathbb{R}^n \to \mathbb{R}$ be convex functions on a convex compact set $X \subseteq \mathbb{R}^n$. Let $\bar{g} : \mathbb{R}^n \to \mathbb{R}$ and $\bar{h} : \mathbb{R}^n \to \mathbb{R}$ be ϵ-polyhedral underestimators of g and h over X. Let $x^g, x^h \in X$ be optimal solutions to problems $(P_{\bar{g}})$, $(P_{\bar{h}})$; and z^*, z^g, and z^h be the optimal values of the problems (P), $(P_{\bar{g}})$, and $(P_{\bar{h}})$, respectively. Then, x^g and x^h are ϵ-optimal solutions of (P). Moreover, $0 \leq z^* - z^g \leq \epsilon$ and $0 \leq z^h - z^* \leq \epsilon$ hold.*

Proof. Note that $z^g \leq z^*$ holds since $\bar{g}(x) \leq g(x)$ holds for all $x \in X$. Then x^g is an ϵ-solution of (P) since we have $g(x^g) - h(x^g) \leq \bar{g}(x^g) - h(x^g) + \epsilon = z^g + \epsilon \leq z^* + \epsilon$. Note that the first inequality holds as $\bar{g}$ is an ϵ-polyhedral underestimator of g. Moreover, $z^* - z^g \leq \epsilon$ holds since

$$\begin{aligned} z^* - z^g &= \inf_{x\in X}\{g(x) - h(x)\} - \inf_{x\in X}\{\bar{g}(x) - h(x)\} \\ &= -\sup_{x\in X}\{h(x) - g(x)\} + \sup_{x\in X}\{h(x) - \bar{g}(x)\} \\ &\leq \sup_{x\in X}\{g(x) - \bar{g}(x)\} \leq \epsilon, \end{aligned}$$

where the first inequality is by Lemma 3.0.2. On the other hand, $z^* \leq z^h$ holds since $\bar{h}(x) \leq h(x)$ holds for all $x \in X$. Moreover, similar to the previous case, we have

$$\begin{aligned} z^h - z^* &= \inf_{x\in X}\{g(x) - \bar{h}(x)\} - \inf_{x\in X}\{g(x) - h(x)\} \\ &= -\sup_{x\in X}\{\bar{h}(x) - g(x)\} + \sup_{x\in X}\{h(x) - g(x)\} \end{aligned}$$

$$\leq \sup_{x \in X} \{h(x) - \bar{h}(x)\} \leq \epsilon.$$

Finally, x^h is an ϵ-solution since $g(x^h) - h(x^h) \leq g(x^h) - \bar{h}(x^h) = z^h \leq z^* + \epsilon$ holds. □

Theorem 3.0.3 suggests that after computing an ϵ-polyhedral underestimator of g or h, one can directly use the primal or dual methods from [1] to solve the problems $(P_{\bar{g}})$ or $(P_{\bar{h}})$ and find ϵ-solutions to problem (P).

3.1 The Base Algorithm

Now, we describe the proposed algorithm which computes an ϵ-polyhedral underestimator of a convex function g over a box $X \subseteq \mathbb{R}^n$, see Assumption 2.2.1, for any precision level $\epsilon > 0$. The main idea is to approximate the epigraph of g over X. To that end, we define the set to be approximated as

$$\mathcal{C} \coloneqq \operatorname{epi} g \cap (X \times \mathbb{R}) \subseteq \mathbb{R}^{n+1}. \tag{3.1.1}$$

Note that as $X \subseteq \mathbb{R}^n$ is compact, the recession cone of $\mathcal{C}$ is

$$\operatorname{recc} \mathcal{C} = K \coloneqq \{(0, k) \in \mathbb{R}^n \times \mathbb{R} \mid k \geq 0\} = \operatorname{cone}\{e^{n+1}\}. \tag{3.1.2}$$

We start with some $x^0 \in \operatorname{int} X$, and compute an outer approximation of $\mathcal{C}$ as

$$C^0 \coloneqq H(g, x^0) \cap (X \times \mathbb{R}), \tag{3.1.3}$$

where $H(g, x^0)$ is as in Lemma 2.2.3. Clearly, C^0 is a convex polyhedral set and by Lemma 2.2.3, $C^0 \supseteq \mathcal{C}$. By construction, the recession cone of C^0 is also K. Then, we have $C^0 = \operatorname{conv} \operatorname{vert} C^0 + K$. Moreover, using Remark 2.2.4, we also know that

$$g^0(x) \coloneqq g(x^0) + c(x^0)^{\mathsf{T}}(x - x^0), \tag{3.1.4}$$

where $c(x^0) \in \partial g(x^0)$, is an underestimator of g such that $\operatorname{epi} g^0 \cap (X \times \mathbb{R}) = C^0$. See Figure 3.1 for an illustration for $n = 1$.

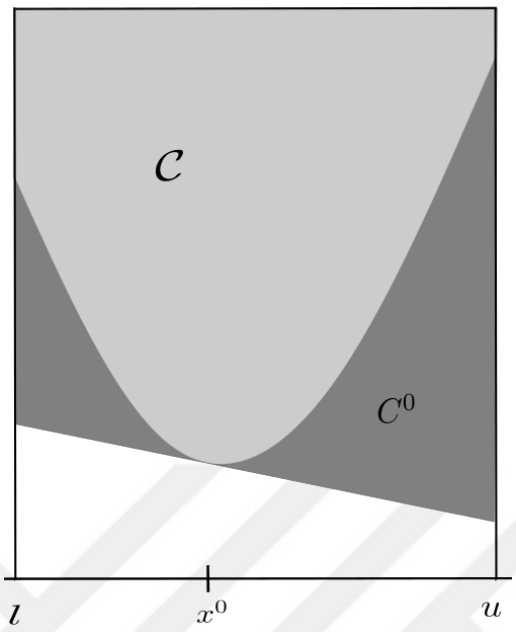


Figure 3.1: The set $\mathcal{C}$ and its initial outer approximation C^0.

At iteration k the algorithm computes $\max_{x \in X}(g(x) - g^k(x))$. Since g^k is a polyhedral convex function, by [1, Corollary 10], an optimal solution exists among the vertices of its epigraph. By the construction of the algorithm, we have $\operatorname{epi} g^k \cap (X \times \mathbb{R}) = C^k$ for every k. Hence an optimal solution $\bar{x}$ is computed as $(\bar{x}, \bar{y}) \in \arg\max_{(x,y) \in \operatorname{vert} C^k}(g(x) - y)$. The algorithm stops if $g(\bar{x}) - g^k(\bar{x}) \leq \epsilon$, and returns g^k and $\operatorname{vert} C^k$. Otherwise, a supporting halfspace $H(g, \bar{x})$ to $\operatorname{epi} g$ at $(\bar{x}, g(\bar{x}))$ is generated. The current outer approximation of $\mathcal{C}$ and the polyhedral underestimator of g are updated accordingly. See Algorithm 1 for the details.

Algorithm 1 Algorithm to compute an ϵ-polyhedral underestimator of a convex function.

1: **Input:** $g : \mathbb{R}^n \to \mathbb{R}, X = [\ell, u] \subseteq \mathbb{R}^n, \epsilon > 0$.
2: Set $k = 0$;
3: $x^0 := \frac{\ell+u}{2}$, set C^0 and g^0 as in (3.1.3) and (3.1.4), respectively and let $\mathcal{S} = \{g^0(x)\}$;
4: **while true do**
5: Compute $\operatorname{vert} C^k$;
6: Let $(\bar{x}, \bar{y}) \in \arg\max_{(x,y)\in \operatorname{vert} C^k}(g(x) - y)$;
7: **if** $g(\bar{x}) - \bar{y} > \epsilon$ **then**
8: $C^{k+1} := C^k \cap H(g, \bar{x})$;
9: $\bar{s}(x) = g(\bar{x}) + c(\bar{x})^{\mathsf{T}}(x - \bar{x})$, $\mathcal{S} \leftarrow \mathcal{S} \cup \{\bar{s}(x)\}$;
10: $g^{k+1}(x) \leftarrow \max\{g^k(x), \bar{s}(x)\}$;
11: $k \leftarrow k + 1$;
12: **else**
13: break;
14: **end if**
15: **end while**
16: **return** $\begin{cases} g^k : \text{an } \epsilon\text{-polyhedral underestimator of } g. \\ \operatorname{vert} C^k : \text{vertices of } \operatorname{epi} g^k \cap (X \times \mathbb{R}). \end{cases}$

The next theorem states that when Algorithm 1 terminates, it returns an ϵ-polyhedral underestimator of g.

Theorem 3.1.1. *Let $g : \mathbb{R}^n \to \mathbb{R}$ be convex and $\epsilon > 0$. When Algorithm 1 stops, it returns an ϵ-polyhedral underestimator of g on X.*

Proof. By Lemma 2.2.3, $C^0 \supseteq \mathcal{C}$. Moreover, by construction, C^0 is a closed convex set and it doesn't include a line. Then, it has at least one vertex by [39, Corollary 18.5.3], that is, we have $\operatorname{vert} C^0 \neq \emptyset$. Similarly, at iteration k, $\operatorname{vert} C^k \neq \emptyset$. For vertex $(\bar{x}, \bar{y}) \in \operatorname{vert} C^k$, a supporting halfspace $H(g, \bar{x})$ to $\operatorname{epi} g$ at $(\bar{x}, g(\bar{x})) \in \operatorname{bd} \mathcal{C}$ is generated, if it is not sufficiently close to the epigraph of g. By Lemma 2.2.3, $H(g, \bar{x}) \supseteq \mathcal{C}$. Since $C^0 \supseteq \mathcal{C}$ and $C^{k+1} = C^k \cap H(g, \bar{x})$ for all $k \geq 0$, we have $C^k \supseteq \mathcal{C}$ for all $k \geq 0$ through the algorithm.

By Remark 2.2.4, g^0 is an affine underestimator of g. Similarly, each $\bar{s}$ introduced in line 9 of Algorithm 1 is an affine underestimator of g. Then, by construction, for any $k \geq 0$ through the algorithm, g^k is a polyhedral underestimator of g. Moreover, we have $C^k = \operatorname{epi} g^k \cap (X \times \mathbb{R})$, in particular, for every $(\bar{x}, \bar{y}) \in \operatorname{vert} C^k$, we have $\bar{y} = g^k(\bar{x})$. Moreover, for every $\bar{x} \in X$, $(\bar{x}, g^k(\bar{x})) \in \operatorname{bd} C^k$.

Assume that Algorithm 1 stops and returns $g^{\bar{k}}$ for some $\bar{k} \geq 1$. We will show that $g^{\bar{k}}$ is an ϵ-polyhedral underestimator of g on X. Let $(\bar{x}, g^{\bar{k}}(\bar{x})) \in \operatorname{vert} C^{\bar{k}}$ be the farthest vertical distance vertex from $\mathcal{C}$ as in line 6 of Algorithm 1. Then $g(\bar{x}) - g^{\bar{k}}(\bar{x}) \leq \epsilon$ holds upon termination. For any $\tilde{x} \in X$, we have $(\tilde{x}, g^{\bar{k}}(\tilde{x})) \in \operatorname{bd} C^{\bar{k}}$. Then,

$$\begin{aligned} g(\tilde{x}) - g^{\bar{k}}(\tilde{x}) &\leq \max_{x \in X} (g(x) - g^{\bar{k}}(x)) \\ &= \max_{(x,y) \in \operatorname{vert} C^{\bar{k}}} (g(x) - y) \\ &= g(\bar{x}) - g^{\bar{k}}(\bar{x}) \leq \epsilon, \end{aligned}$$

where the first equality is by Corollary 2.1.2. □

3.2 A Modified Algorithm

In this section, we describe a modified version of Algorithm 1. The motivation for this variant is to possibly reduce the computational time. To compute the vertices of a polyhedral set, we solve a vertex enumeration problem, which is computationally expensive in general. In Algorithm 1, this is done after finding one supporting halfspace to $\operatorname{epi} g$, which is computed with respect to the 'farthest vertical distance' vertex of the current polyhedral underestimator of $\operatorname{epi} g$.

The initialization of the modified algorithm is exactly as in Algorithm 1. However, at iteration k, Algorithm 2 considers the set of all vertices of the current underestimator C^k. If a vertex $(\bar{x}, \bar{y}) \in \mathbb{R}^n \times \mathbb{R}$ of C^k is sufficiently close to the epigraph of g, it is added to set $\mathcal{V}$, which stores the set of sufficiently close vertices. Otherwise, a supporting halfspace to $\operatorname{epi} g$ at $(\bar{x}, g(\bar{x}))$ is generated and stored. The current polyhedral underestimator of $\mathcal{C}$ is updated by intersecting it with all these supporting halfspaces at once. The polyhedral underestimator of g is updated, accordingly. The algorithm terminates when all the vertices of C^k are close to $\operatorname{epi} g$, see Algorithm 2 for the details. An iteration of Algorithms 1 and 2 is illustrated in Figure 3.2.

Algorithm 2 Algorithm to compute an ϵ-polyhedral underestimator of a convex function.

Input: $g : \mathbb{R}^n \to \mathbb{R}, X = [\ell, u] \subseteq \mathbb{R}^n, \epsilon > 0$.
Set $\mathcal{V} = \emptyset, k = 0, R = \emptyset$;
$x^0 := \frac{\ell+u}{2}$, set C^0 and g^0 as in (3.1.3) and (3.1.4), respectively and let $\mathcal{S} = \{g^0(x)\}$;
while $R \neq \mathbb{R}^{n+1}$ **do**
  Compute $\operatorname{vert} C^k$;
  $R = \mathbb{R}^{n+1}$, $j = 0$, $g^{k,j}(x) := g^k(x)$;
  for all $(\bar{x}, \bar{y}) \in \operatorname{vert} C^k \setminus \mathcal{V}$ **do**
    if $g(\bar{x}) - \bar{y} > \epsilon$ **then**
      $R \leftarrow R \cap H(g, \bar{x})$;
      $\bar{s}(x) = g(\bar{x}) + c(\bar{x})^{\mathsf{T}}(x - \bar{x})$, $\mathcal{S} \leftarrow \mathcal{S} \cup \{\bar{s}(x)\}$;
      $g^{k,j+1}(x) \leftarrow \max\{g^{k,j}(x), \bar{s}(x)\}$, $j \leftarrow j + 1$;
    else
      $\mathcal{V} \leftarrow \mathcal{V} \cup \{\bar{x}\}$;
    end if
  end for
  $C^{k+1} := C^k \cap R$, $g^{k+1}(x) \leftarrow g^{k,j}(x)$, $J^k := j$, $k \leftarrow k + 1$;
end while
return $\begin{cases} g^k : \text{an } \epsilon\text{-polyhedral underestimator of } g. \\ \operatorname{vert} C^k : \text{vertices of } \operatorname{epi} g^k \cap (X \times \mathbb{R}). \end{cases}$

The next theorem states that when Algorithm 2 terminates, it returns an ϵ-polyhedral underestimator of g.

Theorem 3.2.1. *Let $g : \mathbb{R}^n \to \mathbb{R}$ be convex and $\epsilon > 0$. When Algorithm 2 stops, it returns an ϵ-polyhedral underestimator of g on X.*

Proof. For any $k \geq 0$, $\operatorname{vert} C^k \neq \emptyset$, g^k is a polyhedral underestimator of g, and $C^k = \operatorname{epi} g^k \cap (X \times \mathbb{R})$ holds. Assume that Algorithm 2 stops and returns $g^{\bar{k}}$ for some $\bar{k} \geq 1$. This implies that each $(\bar{x}, g^{\bar{k}}(\bar{x})) \in \operatorname{vert} C^{\bar{k}}$ is also an element of $\mathcal{V}$, that is, $\operatorname{vert} C^{\bar{k}} \subseteq \mathcal{V}$. We will show that $g^{\bar{k}}$ is an ϵ-polyhedral underestimator of g on X. Indeed, if $(\bar{x}, g^{\bar{k}}(\bar{x})) \in \operatorname{vert} C^{\bar{k}}$, then $g(\bar{x}) - g^{\bar{k}}(\bar{x}) \leq \epsilon$ holds as $\operatorname{vert} C^{\bar{k}} \subseteq \mathcal{V}$. In general, for $\bar{x} \in X$, we have $(\bar{x}, g^{\bar{k}}(\bar{x})) \in \operatorname{bd} C^{\bar{k}}$, and there exists $\bar{s} \in \mathcal{S}$ with $g^{\bar{k}}(\bar{x}) = \bar{s}(\bar{x})$. Moreover, there exists $K \in \mathbb{N}$, $(\bar{x}_i, \bar{s}(\bar{x}_i)) \in \operatorname{vert} C^{\bar{k}}$ and $\lambda_i \in [0, 1]$ for $i = 1, \ldots, K$, such that $\bar{x} = \sum_{i=1}^K \lambda_i \bar{x}_i$ and $\sum_{i=1}^K \lambda_i = 1$. Then,

$$g(\bar{x}) - g^{\bar{k}}(\bar{x}) = g(\sum_{i=1}^K \lambda_i \bar{x}_i) - \bar{s}(\sum_{i=1}^K \lambda_i \bar{x}_i)$$

$$\leq \sum_{i=1}^{K} \lambda_i g(\bar{x}_i) - \sum_{i=1}^{K} \lambda_i \bar{s}(\bar{x}_i)$$
$$= \sum_{i=1}^{K} \lambda_i (g(\bar{x}_i) - \bar{s}(\bar{x}_i)) \leq \epsilon,$$

where the first inequality is by the convexity of g and affinity of $\bar{s}$ and the last inequality is by $(\bar{x}_i, \bar{s}(\bar{x}_i)) \in \operatorname{vert} C^{\bar{k}} \subseteq \mathcal{V}$. □

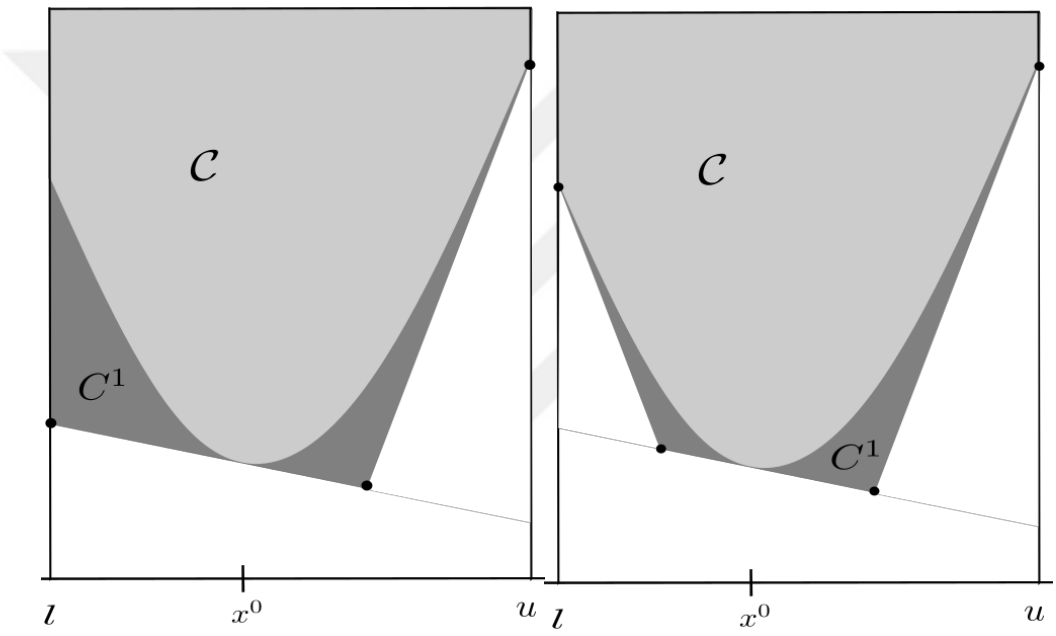


Figure 3.2: The outer approximation C^1 of $\mathcal{C}$ after first iteration of Algorithm 1 (left) and Algorithm 2 (right). $\operatorname{vert} C^1$ are indicated by $\bullet$.

Chapter 4

Convergence Analysis of Algorithms 1 and 2

In this chapter, we determine the convergence rate of Algorithm 1 using some results from the literature [33, 34, 40]. For Algorithm 2, we show that it terminates after fintely many iterations. In case of the latter, we do not establish its convergence rate as the results from [33, 34, 40] do not hold for Algorithm 2.

4.1 Convergence Analysis of Algorithm 1

In this section, we study the convergence of Algorithm 1. For the main results of this section, we assume that Algorithm 1 is run for a closed proper convex function $g : \mathbb{R}^n \to \mathbb{R}$. Moreover, we assume that g is non-polyhedral and the algorithm is run with $\epsilon = 0$.

To establish the convergence rate, we use convergence results of a method for approximating convex compact sets from [33]. For a compact convex set $\mathcal{A} \subseteq \mathbb{R}^{n+1}$, a sequence of outer approximating polytopes $\mathcal{A}_k$, $k \geq 0$ satisfying $\mathcal{A}_0 \supseteq \mathcal{A}_1 \supseteq \ldots \mathcal{A}$ is said to be generated by a *cutting method* if

1. $\mathcal{A}_0 \supseteq \mathcal{A}$ is a polyhedral set which is intersection of supporting halfspaces of $\mathcal{A}$; and

2. $\mathcal{A}_{k+1} = \mathcal{A}_k \cap H_k$ for all $k \geq 0$, where H_k is a supporting halfspace of $\mathcal{A}$.

The following definition and theorem from [33] will be useful in establishing the convergence rate of Algorithm 1.

Definition 4.1.1. *[33, Definition 8.3] Let $\mathcal{A} \subseteq \mathbb{R}^{n+1}$ be compact convex set and $\mathcal{A}_k$, $k \geq 0$ be generated by a cutting method. $(\mathcal{A}_k)_{k\geq 0}$ is called an $H(\eta, \mathcal{A})$-sequence if there exists a constant $\eta > 0$ such that for any $k \geq 0$ it holds that*

$$\delta^H(\mathcal{A}_k, \mathcal{A}_{k+1}) \geq \eta \delta^H(\mathcal{A}_k, \mathcal{A}).$$

Theorem 4.1.2. *[33, Theorems 8.5, 8.6] Let $\eta > 0$, $\mathcal{A} \subseteq \mathbb{R}^{n+1}$ be a convex compact set and $(\mathcal{A}_k)_{k\geq 0}$ be an $H(\eta, \mathcal{A})$-sequence. Then for any $0 < \epsilon < 1$, there exists $N \in \mathbb{N}$ such that for $k \geq N$ it holds that*

$$\delta^H(\mathcal{A}_k, \mathcal{A}) \leq (1+\epsilon)\lambda(\eta) k^{-\frac{1}{n}}.$$

Here, the value of $\lambda(\eta)$ depends on the topological properties of the set $\mathcal{A}$, found in [34, Theorem 2]. In particular, $\lim_{k\to\infty} \delta^H(\mathcal{A}_k, \mathcal{A}) = 0$ holds.

For the convergence rate of Algorithm 1, we work with a convex compact subset A of $\mathbb{R}^{n+1}$ which satisfy $A + K = \mathcal{C}$, where K is the upward cone given as in (3.1.2). To this end, consider the halfspace given by

$$S := \{(x^\mathsf{T}, t)^\mathsf{T} \in \mathbb{R}^n \times \mathbb{R} \mid t \leq b\}, \tag{4.1.1}$$

where $b \coloneqq \sup_{x\in X} g(x)$. Note that $b \in \mathbb{R}$ as $X \subseteq \mathbb{R}^n$ is compact and g is continuous. Now, Lemma 4.1.3 shows that the set

$$A := \operatorname{epi} g \cap (X \times \mathbb{R}) \cap S = \mathcal{C} \cap S \tag{4.1.2}$$

satisfy the required properties.

Lemma 4.1.3. *Let $X \subseteq \mathbb{R}^n$ be convex and compact, $g : X \to \mathbb{R}$ is convex, and sets $S, A \subseteq \mathbb{R}^{n+1}$ be as defined in (4.1.1), (4.1.2), respectively. It holds true that A is a convex compact set satisfying $A + K = \mathcal{C}$.*

Proof. Note that A is closed as it is intersection of closed sets. Moreover, $A \subseteq X \times [a, b]$, where

$a \coloneqq \inf_{x\in X} g(x)$, and $b = \sup_{x\in X} g(x)$. As X is compact, $a, b \in \mathbb{R}$. These imply that A is also bounded, hence compact. Now, let $(x,t) \in A$ and $k \geq 0$. Clearly, $(x, t+k) \in \operatorname{epi} g \cap (X \times \mathbb{R})$, which shows $A + K \subseteq \mathcal{C}$. On the other hand, for $(x,t) \in \operatorname{epi} g$ with $x \in X$, we have $(x,t) = (x, g(x)) + (0, t - g(x)) \in A + K$, since $g(x) \leq b$ and $t - g(x) \geq 0$. □

We also define the following sets

$$A^k := \operatorname{epi} g^k \cap (X \times \mathbb{R}) \cap S = C^k \cap S, \tag{4.1.3}$$

where g^k for $k \geq 0$ are as in Algorithm 1. By Lemma 4.1.3, these are convex compact sets satisfying $A^{(\cdot)} + K = \operatorname{epi} g^{(\cdot)} \cap (X \times \mathbb{R})$. Moreover,

$$A \subseteq A^{k+1} \subseteq A^k \tag{4.1.4}$$

holds for all $k \geq 0$.

Remark 4.1.4. *A simple but important observation regarding the sets A^k is that* $\operatorname{vert} A^k = \operatorname{vert} C^k \cup (\operatorname{vert} X \times \{b\})$ *for all $k \geq 0$. Moreover,* $\operatorname{vert} X \times \{b\} \subseteq A$. *This implies that for any $k \geq 0$, we have*

$$\max_{(x,y)\in \operatorname{vert} C^k} (g(x) - y) = \max_{(x,y)\in \operatorname{vert} A^k} (g(x) - y).$$

If the maximum is positive, then the arguments of the maxima are equal as well. See Figure 4.1 for an illustration with $n = 1$.

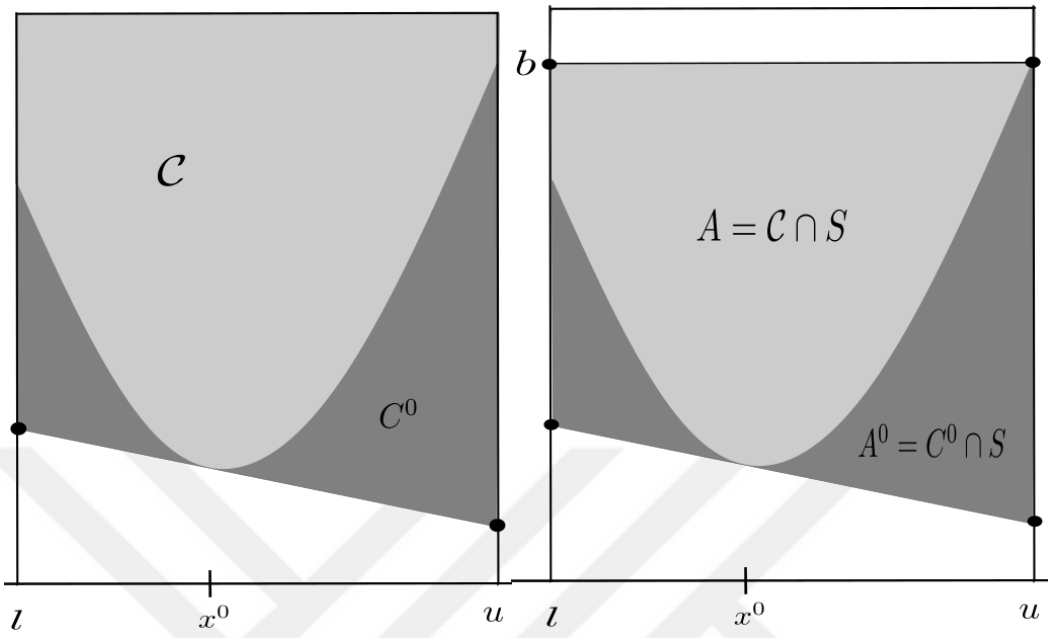


Figure 4.1: An illustration of the sets $\mathcal{C}, C^0$ (left) and A, A^0 (right). $\operatorname{vert} C^0$ and $\operatorname{vert} A^0$ are indicated by $\bullet$.

Lemma 4.1.5. *Let $k \geq 0$, $(\bar{x}, \bar{y}) \in \arg\max_{(x,y)\in \operatorname{vert} C^k}(g(x) - y)$ and A, A^k be as given in* (4.1.2),(4.1.3)*, respectively. Then, $\delta^H(A^k, A) \leq g(\bar{x}) - \bar{y}$.*

Proof. The statement holds trivially if $g(\bar{x}) - \bar{y} = 0$ since it implies that $A^k = A$. On the other hand, from Lemma 2.1.4 and Remark 4.1.4, we obtain

$$\begin{aligned}
\delta^H(A^k, A) &= \max_{(x,y)\in \operatorname{vert} A^k} \inf_{(x^a,y^a)\in A} \|(x^a, y^a) - (x, y)\| \\
&\leq \max_{(x,y)\in \operatorname{vert} A^k} \|(x, g(x)) - (x, y)\| \\
&= \max_{(x,y)\in \operatorname{vert} A^k} (g(x) - y) \\
&= \max_{(x,y)\in \operatorname{vert} C^k} (g(x) - y) = g(\bar{x}) - \bar{y}. \qquad (4.1.5)
\end{aligned}$$

□

Theorem 4.1.6. *Assume $g : \mathbb{R}^n \to \mathbb{R}$ is a closed proper convex. Let $A, (A^k)_{k\geq 0}$ be as given in* (4.1.2)*,* (4.1.3)*, respectively. $(A^k)_{k\geq 0}$ is an $H(\eta, A)$-sequence for some $\eta > 0$, that is, it satisfies $\delta^H(A^k, A^{k+1}) \geq \eta\delta^H(A^k, A)$ for all $k \geq 0$.*

Proof. Let $k \geq 0$ be arbitrary and $(\bar{x}, \bar{y}) \in \arg\max_{(x,y)\in \operatorname{vert} A^k}(g(x) - y)$. By Remark 4.1.4, $(\bar{x}, \bar{y}) \in \arg\max_{(x,y)\in \operatorname{vert} C^k}(g(x) - y)$ and $\bar{y} = g^k(\bar{x})$. Indeed, Algorithm 1 considers $(\bar{x}, \bar{y})$ at k^{th} iteration and $A^{k+1} = A^k \cap H(g, \bar{x})$, where $H(g, \bar{x}) = \{(x, t) \in \mathbb{R}^n \times \mathbb{R} \mid t - c(\bar{x})^\mathsf{T} x \geq g(\bar{x}) - c(\bar{x})^\mathsf{T}\bar{x}\}$ is a supporting halfspace to $\operatorname{epi} g$ at $(\bar{x}, g(\bar{x})) \in A$. Here, $c(\bar{x}) \in \partial g(\bar{x})$ is a subgradient of g at $\bar{x}$, see Lemma 2.2.3. Let $(x', y') \in A^{k+1}$ be arbitrary and $m := \frac{(-c(\bar{x})^\mathsf{T}, 1)^\mathsf{T}}{\left\|(-c(\bar{x})^\mathsf{T}, 1)\right\|_*}$. Then, $m^\mathsf{T}(x', y') \geq m^\mathsf{T}(\bar{x}, g(\bar{x}))$ implies

$$\begin{aligned} m^\mathsf{T}((x', y') - (\bar{x}, \bar{y})) &\geq m^\mathsf{T}((\bar{x}, g(\bar{x})) - (\bar{x}, \bar{y})) \\ &= m^\mathsf{T}(0, g(\bar{x}) - \bar{y}) \\ &= \frac{g(\bar{x}) - \bar{y}}{\|(-c(\bar{x})^\mathsf{T}, 1)\|_*} \geq \frac{\delta^H(A^k, A)}{\|(-c(\bar{x})^\mathsf{T}, 1)\|_*}, \end{aligned}$$

where the last inequality is by Lemma 4.1.5. From Hölder's inequality, we have

$$m^\mathsf{T}((x', y') - (\bar{x}, \bar{y})) \leq \|m\|_* \|(x', y') - (\bar{x}, \bar{y})\| = \|(x', y') - (\bar{x}, \bar{y})\|.$$

Then, $d((\bar{x}, \bar{y}), A^{k+1}) = \inf_{(x',y')\in A^{k+1}} \|(x', y') - (\bar{x}, \bar{y})\| \geq \frac{\delta^H(A^k, A)}{\left\|(-c(\bar{x})^\mathsf{T}, 1)\right\|_*}$. From Lemma 2.1.4, we obtain

$$\delta^H(A^k, A^{k+1}) = \max_{v\in \operatorname{vert} A^k} d(v, A^{k+1}) \geq d((\bar{x}, \bar{y}), A^{k+1}) \geq \frac{\delta^H(A^k, A)}{\|(-c(\bar{x})^\mathsf{T}, 1)\|_*}.$$

From [39, Theorem 24.7], $\bigcup_{x\in X} \partial g(x)$ is a nonempty bounded closed subset. This implies that $\sup_{x\in X, c(x)\in\partial g(x)} \left\|(-c(x)^\mathsf{T}, 1)\right\|_* \leq \frac{1}{\eta}$ for some $\eta > 0$. Hence, the required inequality holds. □

Corollary 4.1.7. *Assume $g : \mathbb{R}^n \to \mathbb{R}$ is a closed proper convex. Let $A, (A^k)_{k\geq 0}$ be as given in* (4.1.2), (4.1.3), *respectively.*

(a) The approximation error for the sequence $(A^k)_{k\geq 0}$ decreases by the order $\mathcal{O}(k^{-\frac{1}{n}})$.

(b) $\lim_{k\to\infty} \delta^H(A^k, A) = 0$.

Proof. (a) By Theorem 4.1.6, $(A^k)_{k\geq 0}$ is an $H(\eta, A)$-sequence for some $\eta > 0$. Then by Theorem 4.1.2, for any $0 < \epsilon < 1$, there exists $N \in \mathbb{N}$ such that for $k \geq N$

$$\delta^H(A^k, A) \leq (1 + \epsilon)\lambda(\eta)k^{-\frac{1}{n}}, \text{ where } \lambda(\eta) \text{ is as given in Theorem 4.1.2.}$$

(b) follows directly from (a). □

Next, we prove that the sequence of polyhedral convex approximations $\{g^k\}_k$ returned by Algorithm 1 converges uniformly to function g. To show this, we use the following theorem by Ulisse Dini. The proof of Dini's theorem can be viewed in [41, Chapter 12, Theorem 12.1].

Theorem 4.1.8. *[41, Chapter 12, Theorem 12.1] Let X be a compact set and let $(f_n)_{n\in\mathbb{N}}$ be a monotone sequence of continuous real-valued functions on X which converges pointwise to a continuous function $f : X \to \mathbb{R}$. Then the convergence is uniform.*

Proposition 4.1.9. *Let $\epsilon = 0$ in Algorithm 1 and $\{g^k\}_{k\geq 0}$ be a sequence of continuous real-valued polyhedral convex approximations on X outputted by the algorithm. Then $\{g^k\}_{k\geq 0}$ converges uniformly to g.*

Proof. By Corollary 4.1.7, for any $\epsilon = \tilde{\epsilon} > 0$, there exists a $K \in \mathbb{N}$ such that $g(x) - g^k(x) \leq \tilde{\epsilon}$ for all $k \geq K$. Thus, the sequence $\{g^k\}_{k\geq 0}$ converges pointwise to g. Indeed this is the classical definition of pointwise convergence. Since X is compact, by Theorem 4.1.8, the sequence also converges uniformly to g. □

4.2 Finiteness of Algorithm 2

In this section, we prove that Algorithm 2 is finite. Let $\mathcal{C}, S, A$, and $(A^k)_{k\geq 0}$ be the sets defined in (3.1.1), (4.1.1) , (4.1.2), and (4.1.3) and K be the cone as defined in (3.1.2). Recall from Lemma 4.1.3, that A, A^k are convex compact sets in $\mathbb{R}^{n+1}$ satisfying $A^{(\cdot)} + K = \operatorname{epi} g^{(\cdot)} \cap (X \times \mathbb{R})$. Moreover, recall from (4.1.4), that $A \subseteq A^{k+1} \subseteq A^k$ holds for all $k \geq 0$. For the finiteness of Algorithm 2, first we form a compact set $A^0 + \mathbb{B}[0, \frac{\epsilon}{2\beta}] \subseteq \mathbb{R}^{n+1}$. Then, we show that a sequence of subsets can be constructed through the iterations of Algorithm 2. Finiteness follows as these subsets are non-intersecting and they have same volume. For the main results of this section, we assume that Algorithm 2 is run for a closed proper convex function $g : \mathbb{R}^n \to \mathbb{R}$.

Remark 4.2.1. *Recall that $\bigcup_{x\in X} \partial g(x)$ is a nonempty bounded closed subset by [39, Theorem 24.7]. Then, $\beta \coloneqq \sup_{x\in X, c(x)\in\partial g(x)} \left\|(-c(x)^\mathsf{T}, 1)\right\|_*$ is well defined and $\beta > 0$.*

Next, we provide two useful results before proving the finiteness of Algorithm 2.

Lemma 4.2.2. *Fix $\epsilon > 0$. Let $\bar{v} = (\bar{x}, \bar{g}(\bar{x})) \notin A$, where $\bar{g}$ is a polyhedral underestimator of g, and $H^\epsilon(g, \bar{x})$ be a halfspace defined by*

$$H^\epsilon(g, \bar{x}) \coloneqq \{s \in \mathbb{R}^{n+1} \mid m^\mathsf{T} s \geq m^\mathsf{T} \bar{a} - \frac{\epsilon}{2\beta}\}, \tag{4.2.1}$$

where β is as defined in Remark 4.2.1, $m \coloneqq \frac{(-c(\bar{x})^\mathsf{T}, 1)^\mathsf{T}}{\left\|(-c(\bar{x})^\mathsf{T}, 1)\right\|_*}$, and $\bar{a} = (\bar{x}, g(\bar{x}))$. If $g(\bar{x}) - \bar{g}(\bar{x}) > \epsilon$, then $\mathbb{B}[\bar{v}, \frac{\epsilon}{4\beta}] \cap H^\epsilon(g, \bar{x}) = \emptyset$.

Proof. Let $s^* \in H^\epsilon(g, \bar{x})$ be arbitrary. From the definition of $H^\epsilon(g, \bar{x})$, we have $m^\mathsf{T} s^* \geq m^\mathsf{T}(\bar{v} + (\bar{a} - \bar{v})) - \frac{\epsilon}{2\beta} = m^\mathsf{T}\bar{v} + \frac{g(\bar{x}) - \bar{g}(\bar{x})}{\left\|(-c(\bar{x})^\mathsf{T}, 1)\right\|_*} - \frac{\epsilon}{2\beta} \geq m^\mathsf{T}\bar{v} + \frac{g(\bar{x}) - \bar{g}(\bar{x})}{\beta} - \frac{\epsilon}{2\beta}$. Equivalently,

$$m^\mathsf{T}(s^* - \bar{v}) \geq \frac{g(\bar{x}) - \bar{g}(\bar{x})}{\beta} - \frac{\epsilon}{2\beta}. \tag{4.2.2}$$

From Hölder's inequality, we have

$$|m^\mathsf{T}(s^* - \bar{v})| \leq \|m\|_* \|s^* - \bar{v}\| = \|s^* - \bar{v}\|. \tag{4.2.3}$$

If $g(\bar{x}) - \bar{g}(\bar{x}) > \epsilon$, then using (4.2.2) and (4.2.3), we obtain

$$\|s^* - \bar{v}\| \geq |m^\mathsf{T}(s^* - \bar{v})| \geq \frac{g(\bar{x}) - \bar{g}(\bar{x})}{\beta} - \frac{\epsilon}{2\beta} > \frac{\epsilon}{\beta} - \frac{\epsilon}{2\beta} = \frac{\epsilon}{2\beta}.$$

Therefore, $s^* \notin \mathbb{B}[\bar{v}, \frac{\epsilon}{4\beta}]$, which implies $\mathbb{B}[\bar{v}, \frac{\epsilon}{4\beta}] \cap H^\epsilon(g, \bar{x}) = \emptyset$.

□

The following lemma is an observation that can be found also in [32, Lemma 2.1]. Here it is restated in terms of the terminology used here. It will be useful in proving the finiteness of Algorithm 2.

Lemma 4.2.3. *Let $\bar{v} = (\bar{x}, \bar{g}(\bar{x})) \notin A$ and $H(g, \bar{x})$ and $H^\epsilon(g, \bar{x})$ be halfspaces defined by Lemma 2.2.3 and (4.2.1), respectively. Then $H(g, \bar{x}) + \mathbb{B}[0, \frac{\epsilon}{2\beta}] \subseteq H^\epsilon(g, \bar{x})$.*

Proof. Let $s \in H(g, \bar{x}), s^* \in \mathbb{B}[0, \frac{\epsilon}{2\beta}]$. Assume to the contrary that $s + s^* \notin H^\epsilon(g, \bar{x})$, then

$m^\mathsf{T}(s+s^*) < m^\mathsf{T}\bar{a} - \frac{\epsilon}{2\beta}$. Since, $s \in H(g, \bar{x})$,

$$m^\mathsf{T}\bar{a} + m^\mathsf{T}s^* \leq m^\mathsf{T}(s+s^*) < m^\mathsf{T}\bar{a} - \frac{\epsilon}{2\beta}.$$

This gives $m^\mathsf{T}s^* < -\frac{\epsilon}{2\beta}$. Then $|m^\mathsf{T}s^*| > \frac{\epsilon}{2\beta}$. Using Hölder's inequality, we get

$$\frac{\epsilon}{2\beta} < |m^\mathsf{T}s^*| \leq \|m\|_* \|s^*\| \leq \|m\|_* \frac{\epsilon}{2\beta},$$

where the second inequality comes from $s^* \in \mathbb{B}[0, \frac{\epsilon}{2\beta}]$. This gives $\|m\|_* > 1$, which is a contradiction since m is a unit vector. □

Theorem 4.2.4. *Let $\epsilon > 0$. Algorithm 2 stops after finitely many iterations.*

Proof. The set A^k has finite number of vertices for every $k \geq 0$. It suffices to prove that there exists a $k_\epsilon \geq 0$ such that for every vertex $v = (x, g^{k_\epsilon}(x)) \in \operatorname{vert} A^{k_\epsilon}$, we have $g(x) - g^{k_\epsilon}(x) \leq \epsilon$. Assume to the contrary that for every $k \geq 0$, there is a vertex $v^k \in \operatorname{vert} A^k$ such that $g(x^k) - g^k(x^k) > \epsilon$. For the rest of the proof, we fix an arbitrary $v^k \in \operatorname{vert} A^k$ satisfying this condition.

By Lemma 4.1.3, A^0 is a compact set, and the closed ball $\mathbb{B}[0, \frac{\epsilon}{2\beta}]$ is also compact. Hence $A^0 + \mathbb{B}[0, \frac{\epsilon}{2\beta}]$ is compact by [42, Lemma 5.3]. Define for an arbitrary $k \geq 0$, $\mathcal{B}^k := \mathbb{B}[v^k, \frac{\epsilon}{4\beta}] = \{v^k\} + \mathbb{B}[0, \frac{\epsilon}{4\beta}]$. Next, we show that $\mathcal{B}^k \subseteq A^0 + \mathbb{B}[0, \frac{\epsilon}{2\beta}]$. Since $v^k \in A^k$

$$\mathcal{B}^k \subseteq \{v^k\} + \mathbb{B}[0, \frac{\epsilon}{2\beta}] \subseteq A^k + \mathbb{B}[0, \frac{\epsilon}{2\beta}] \subseteq A^0 + \mathbb{B}[0, \frac{\epsilon}{2\beta}]. \tag{4.2.4}$$

Hence, $\mathcal{B}^k \subseteq A^0 + \mathbb{B}[0, \frac{\epsilon}{2\beta}]$.

To prove $\mathcal{B}^m \cap \mathcal{B}^n = \emptyset$, for every $m, n \geq 0$ with $m \neq n$ without loss of generality assume that $m < n$. By (4.1.4), $A^n \subseteq A^{m+1}$. From Lemma 4.2.2, we have $\mathcal{B}^m \cap H^\epsilon(g, x^m) = \emptyset$. Moreover, we have

$$A^n + \mathbb{B}[0, \frac{\epsilon}{2\beta}] \subseteq A^{m+1} + \mathbb{B}[0, \frac{\epsilon}{2\beta}] = (A^m \cap \bigcap_{\substack{v \in \operatorname{vert} A^m \\ g(x^v) - g^m(x^v) > \epsilon}} H(g, x^v)) + \mathbb{B}[0, \frac{\epsilon}{2\beta}] \subseteq H(g, x^m) + \mathbb{B}[0, \frac{\epsilon}{2\beta}], \tag{4.2.5}$$

where $H(g, x^m)$ is a supporting halfspace to epi g at $(x^m, g(x^m))$ as obtained in Lemma 2.2.3.

Using Lemma 4.2.3 with (4.2.5), we get

$$A^n + \mathbb{B}[0, \frac{\epsilon}{2\beta}] \subseteq H(g, x^m) + \mathbb{B}[0, \frac{\epsilon}{2\beta}] \subseteq H^\epsilon(g, x^m).$$

This implies that $\mathcal{B}^m \cap (A^n + \mathbb{B}[0, \frac{\epsilon}{2\beta}]) = \emptyset$. Moreover, $\mathcal{B}^n \subseteq A^n + \mathbb{B}[0, \frac{\epsilon}{2\beta}]$ from (4.2.4). Hence $\mathcal{B}^m \cap \mathcal{B}^n = \emptyset$. This is a contradiction as these imply that there is an infinite number of non-intersecting sets, with the same positive volume, contained in the compact set $A^0 + \mathbb{B}[0, \frac{\epsilon}{2\beta}]$. □

Corollary 4.2.5. *Let $\epsilon = 0$ in Algorithm 2 and $\{g^k\}_{k\geq 0}$ be a sequence of continuous real-valued polyhedral convex approximations on X outputted by the algorithm. Then $\{g^k\}_{k\geq 0}$ converges uniformly to g.*

Proof. By Theorem 4.2.4, Algorithm 2 is finite if it is called with any $\epsilon = \tilde{\epsilon} > 0$. Then, there exists $K \in \mathbb{N}$ such that $g(x) - g^k(x) \leq \tilde{\epsilon}$ for all $k \geq K$. Thus, the sequence $\{g^k\}_{k\geq 0}$ converges pointwise to g. Indeed this is the classical definition of pointwise convergence. Since X is compact, by Theorem 4.1.8, the sequence also converges uniformly to g. □

Chapter 5

An Algorithm for solving DC Problems

The solution methodology proposed in Chapter 3 is a naive approach, which uses the existing exact solution algorithm from [1] for DC programming problems involving at least one polyhedral convex function. To that end, Algorithms 1 and 2 return an ϵ-polyhedral underestimator of a given convex function over a convex compact set X. In this section, we propose an alternative algorithm to solve the general DC programming problems more directly. Even though the general idea is, in a way, similar to Algorithm 1, this algorithm does not compute an ϵ-polyhedral underestimator of the convex function g over the feasible set X. Instead, it keeps updating the underestimator locally while looking for an ϵ-solution of the DC problem directly. When Algorithm 3 terminates, the resulting underestimator is not necessarily an ϵ-polyhedral underestimator.

5.1 A DC Algorithm (Algorithm 3)

Considering problem (P), let $\bar{g} : \mathbb{R}^n \to \mathbb{R}$ be an underestimator of g. Recall that $(P_{\bar{g}})$ is given by $\min_{x \in X}(\bar{g}(x) - h(x))$. The next theorem will be helpful in explaining the working mechanism of the algorithm.

Theorem 5.1.1. *Let $\epsilon > 0$, $g : \mathbb{R}^n \to \mathbb{R}$, and $h : \mathbb{R}^n \to \mathbb{R}$ be convex functions on a convex*

compact set $X \subseteq \mathbb{R}^n$. Let $\bar{g} : \mathbb{R}^n \to \mathbb{R}$ be a polyhedral underestimator of g over X. Let $\bar{x} \in X$ be an optimal solution to the problem $(P_{\bar{g}})$, and z^*, $\bar{z}$ be the optimal values of the problems (P),$(P_{\bar{g}})$, respectively. If $g(\bar{x}) - \bar{g}(\bar{x}) \leq \epsilon$, then $\bar{x}$ is an ϵ-optimal solution of (P). Moreover, $0 \leq z^* - \bar{z} \leq \epsilon$ holds.

Proof. Note that $\bar{z} \leq z^*$ holds since $\bar{g}(x) \leq g(x)$ holds for all $x \in X$. Then, $\bar{x}$ is an ϵ-solution of (P) since $g(\bar{x}) - h(\bar{x}) \leq \bar{g}(\bar{x}) - h(\bar{x}) + \epsilon = \bar{z} + \epsilon \leq z^* + \epsilon$. This also implies that $g(\bar{x}) - h(\bar{x}) - \bar{z} \leq \epsilon$. Then, as z^* is the optimal value of (P), we obtain, $z^* - \bar{z} \leq g(\bar{x}) - h(\bar{x}) - \bar{z} \leq \epsilon$. □

As Theorem 5.1.1 suggests, in Algorithm 3, the aim is to generate a polyhedral underestimator $\bar{g}$ of the function g, such that $g(\bar{x}) - \bar{g}(\bar{x}) \leq \epsilon$, where $\bar{x}$ solves $(P_{\bar{g}})$ optimally. To that end, we use some terminology as exactly they are used in Chapter 3. In particular, let $\mathcal{C}$ be as given in (3.1.1). Moreover, the initialization of Algorithm 3 is also the same as in Algorithm 1. In particular, we set $\bar{C}^0 \coloneqq C^0$, see (3.1.3), as the initial outer approximation of $\mathcal{C}$. Recall that K is the recession cone of $\mathcal{C}$ and $\bar{C}^0$, see (3.1.2). Moreover, $\bar{C}^0 = \operatorname{epi} g^0 \cap (X \times \mathbb{R})$, where $g^0(x)$ is as in (3.1.4). As it will be explained below, the algorithm iterates by updating the epigraph of the current underestimator so that for each iteration k, $\bar{C}^k = \operatorname{epi} g^k \cap (X \times \mathbb{R})$ holds true.

At iteration k, where $k \geq 1$, the algorithm considers the current underestimator g^{k-1} of the function g and computes an optimal solution to the following problem:

$$\min_{x \in X} (g^{k-1}(x) - h(x)). \tag{P_k}$$

Since g^{k-1} is a polyhedral convex function, the existence of an optimal solution among the vertices of C^{k-1} is guaranteed by [1, Corollary 10]. Hence, an optimal solution x^k can be computed as

$$(x^k, y^k) \in \operatorname*{arg\,min}_{(x,y) \in \operatorname{vert} C^{k-1}} (y - h(x)).$$

Note that $y^k = g^{k-1}(x^k)$ holds by construction. The algorithm checks if $g(x^k) - g^{k-1}(x^k) \leq \epsilon$. If this is the case, x^k is returned. Otherwise, a supporting halfspace to $\operatorname{epi} g$ at $(x^k, g(x^k))$ is generated. The current outer approximation of $\mathcal{C}$ and the polyhedral underestimator of g are updated accordingly, see Algorithm 3 for the details.

Algorithm 3 An algorithm to compute an ϵ-solution to (P)

1: **Input:** Problem (P), $\epsilon > 0$.
2: Set $k = 0$, $x^0 := \frac{l+u}{2}$, set $\bar{C}^0, g^0$ as in (3.1.3),(3.1.4), respectively.
3: **while true do**
4: $k \leftarrow k + 1$;
5: Compute $\operatorname{vert} \bar{C}^{k-1}$, let $(x^k, y^k) \in \arg\min_{(x,y)\in \operatorname{vert} C^{k-1}}(y - h(x))$ (i.e., solve (P_k));
6: **if** $g(x^k) - y^k > \epsilon$ **then**
7: $s^k(x) = g(x^k) + c(x^k)^\mathsf{T}(x - x^k)$;
8: $g^k(x) \leftarrow \max\{g^{k-1}(x), s^k(x)\}$;
9: $\bar{C}^k := \bar{C}^{k-1} \cap H(g, x^k)$;
10: **else**
11: break;
12: **end if**
13: **end while**
14: **return** x^k : ϵ-optimal solution of (P) .

The next theorem states that when Algorithm 3 terminates, it returns an ϵ-optimal solution of (P).

Theorem 5.1.2. *Consider problem (P) and let $\epsilon > 0$. When Algorithm 3 stops, it returns an ϵ-optimal solution of (P).*

Proof. For any $k \geq 0$, $\operatorname{vert} \bar{C}^k \neq \emptyset$, g^k is a polyhedral underestimator of g, and $\bar{C}^k = \operatorname{epi} g^k \cap (X \times \mathbb{R})$ holds. Then, in line 5 of Algorithm 3, the algorithm returns a solution x^k to (P_k) by [1, Corollary 11], where $y^k = g^{k-1}(x^k)$ holds. If the algorithm stops at iteration $\bar{k}$ for some $\bar{k} \geq 1$, then $g(x^{\bar{k}}) - g^{\bar{k}-1}(x^{\bar{k}}) \leq \epsilon$ holds and by Theorem 5.1.1, $x^{\bar{k}}$ is an ϵ-solution to (P). □

5.2 Convergence of Algorithm 3

We study the convergence of Algorithm 3 in this section. In particular, we show that the limit point of the sequence $\{x^k\}_{k\geq 0}$, found by Algorithm 3, is a global minimizer to the DC program (P) if ϵ is set to zero. Let us introduce the following quantities:

$$\begin{aligned} a_k &:= g^{k-1}(x^k) - h(x^k), \quad k = 1, 2, \dots \\ b_k &:= g^k(x^k) - h(x^k), \quad k = 0, 1, 2, \dots \end{aligned} \tag{5.2.1}$$

The following lemma highlights some properties of the functions g^k and the quantities a_k and b_k.

Lemma 5.2.1. *Consider problem* (P)*. Assume* $\epsilon = 0$ *in Algorithm 3. Let* g^k, s^k *be as in Algorithm 3 and* b_k, a_k *be as in* (5.2.1)*. Then,*

(a) $g^k(x^k) = g(x^k) = s^k(x^k)$ *holds for all* $k \in \{1, 2, \ldots\}$,

(b) $a_k \leq \min_{x \in X}(g(x) - h(x)) \leq b_k$ *holds for all* $k \in \{1, 2, \ldots\}$.

Proof. (a) Note that $g(x^k) = s^k(x^k)$ for every $k \geq 1$ by definition of s^k. Moreover, g^k is an underestimator of g for all $k \geq 0$. Then, we have

$$g^k(x^k) \leq g(x^k) = s^k(x^k) \leq \max\{g^{k-1}(x^k), s^k(x^k)\} = g^k(x^k).$$

(b) Since x^k is an optimal solution of (P_k), and g^{k-1} is an underestimator of g, we have

$$a_k = g^{k-1}(x^k) - h(x^k) = \min_{x \in X}(g^{k-1}(x) - h(x)) \leq \min_{x \in X}(g(x) - h(x)) \leq g(x^k) - h(x^k) = b_k,$$

where the last equality is by (a).

□

Theorem 5.2.2. *Consider problem* (P)*. Assume* $\epsilon = 0$ *in Algorithm 3. Every limit point of the sequence* $\{x^k\}_{k \geq 0}$ *outputted by Algorithm 3 is a global minimizer of* (P)*.*

Proof. The compactness of X implies that the limit points of the sequence $\{x^k\}_{k \geq 0}$ exist in X. Let $\{x^{k_j}\}_{j \geq 1}$ be a a convergent subsequence. With the convention that $s^0(x) = g^0(x)$ (used in the second equality below) and using definition of s^i, that is, $s^i(x) = g(x^i) + c(x^i)^\mathsf{T}(x - x^i)$, we have

$$\begin{aligned}
b_{k_{j-1}} &= g^{k_{j-1}}(x^{k_{j-1}}) - h(x^{k_{j-1}}) \\
&= \max_{0 \leq i \leq k_{j-1}} (s^i(x^{k_{j-1}})) - h(x^{k_{j-1}}) \\
&= \max_{0 \leq i \leq k_{j-1}} (s^i(x^{k_j}) + c(x^i)^\mathsf{T}(x^{k_{j-1}} - x^{k_j})) - h(x^{k_{j-1}}) \\
&\leq \max_{0 \leq i \leq k_{j-1}} s^i(x^{k_j}) + \max_{0 \leq i \leq k_{j-1}} \|c(x^i)\| \|x^{k_{j-1}} - x^{k_j}\|_* - h(x^{k_{j-1}})
\end{aligned}$$

$$= g^{k_{j-1}}(x^{k_j}) - h(x^{k_j}) + \max_{0\le i\le k_{j-1}} \|c(x^i)\|\|x^{k_{j-1}} - x^{k_j}\|_* + h(x^{k_j}) - h(x^{k_{j-1}})$$
$$= a_{k_j} + \max_{0\le i\le k_{j-1}} \|c(x^i)\|\|x^{k_{j-1}} - x^{k_j}\|_* + h(x^{k_j}) - h(x^{k_{j-1}}),$$

where the inequality is by the triangle and Hölder inequalities. As h is continuous, $\lim_{j\to\infty}\left(\max_{0\le i\le k_{j-1}} \|c(x^i)\|\|x^{k_{j-1}} - x^{k_j}\|_* + h(x^{k_j}) - h(x^{k_{j-1}})\right) = 0$. Moreover, by Lemma 5.2.1 (b), we have $a_{k_j} \le \min_{x\in X}(g(x) - h(x)) \le b_{k_j}$. Hence, we obtain

$$\limsup_{j\to\infty} b_{k_{j-1}} \le \min_{x\in X}(g(x) - h(x)) \le \liminf_{j\to\infty} b_{k_j}.$$

This shows that

$$\lim_{j\to\infty}(g(x^{k_j}) - h(x^{k_j})) = \lim_{j\to\infty}(g^{k_j}(x^{k_j}) - h(x^{k_j})) = \lim_{j\to\infty} b_{k_j} = \min_{x\in X}(g(x) - h(x)),$$

where we use Lemma 5.2.1 (a) in the first equality. □

Corollary 5.2.3. *Let x^* be the global minimizer of (P) and $z^* = g(x^*) - h(x^*)$. Then, Algorithm 3 stops after finitely many iterations, when ϵ is set to a positive number, $\tilde{\epsilon}$.*

Proof. By Theorem 5.2.2, the sequence $\{x^k\}_{k\ge 0}$ converges to x^* if ϵ is set to zero. Then for any $\tilde{\epsilon} > 0$, there exists $\tilde{K} \in \mathbb{N}$ such that $|g(x^k) - h(x^k) - z^*| \le \tilde{\epsilon}$ for $k \ge \tilde{K}$. Note that if the algorithm is run for $\epsilon = \tilde{\epsilon}$ instead of $\epsilon = 0$, then the first $\tilde{K}$ iterations would be the same by the structure of the algorithm. In particular, without loss of generality, we can assume that the same (x^k, y^k) in line 5 of Algorithm 3 is selected for every $k \le \tilde{K}$. This implies that Algorithm 3 stops in $\tilde{K}$ iterations when it runs with $\epsilon = \tilde{\epsilon}$. □

Chapter 6

Experimental Results

In this chapter, we test few examples to assess the performance of Algorithms 1, 2, and 3 compared with the DCECAM algorithm from [2]. For univariate examples, we also compare the PS algorithm [3, 4, 31, 43]. We provide short descriptions of the algorithms from the literature in Appendix A.

All algorithms are executed using MATLAB R2022a along with *bensolve tools* [44] to run the vertex enumeration function in each iteration, respectively. The tests are run on a computer having a 3.6 GHz Intel Core i7 with a 64 GB RAM.

6.1 Test Examples

We consider eight examples: Examples 6.1.1 and 6.1.2 are univariate examples [2, Problems 10.2, 10.3]. Examples 6.1.3, 6.1.4, 6.1.5, 6.1.6 are bivariate examples [2, Problems 10.1, 10.6, 10.7, 10.8], [45]. Example 6.1.7 is a scalable example that is solved for univariate, bivariate, and trivariate instances [2, Problem 10.5], [46]. Example 6.1.8 has a convex objective function [2, Problem 10.4]. For this example, we use the algorithms designed for DC programs to solve a convex program.

Example 6.1.1. *Consider the example for* $a \in \{0, 1.5, 9\}$:

$$\textit{minimize} \quad f(x) := a\sqrt{|1-x|} + |2-x|^3$$

$$\textit{subject to} \quad 1 \leq x \leq 3.$$

For the DC function f, we have $f = g - h$, where g, h are convex functions given by

$$g(x) = |2 - x|^3, \quad h(x) = -a\sqrt{|1 - x|}.$$

It is not difficult to see that when $a \in \{1.5, 9\}$, the problem attains an optimal solution at $x^* = 1$ with minimum value 1, and when $a = 0$, it attains an optimal solution at $x^* = 2$ with minimum value 0. Indeed, when $a = 0$, the problem becomes a convex problem.

Example 6.1.2. *Consider the example*

$$\begin{aligned} &\textit{minimize} \quad f(x) := -\log(x) + \min\{\sqrt{|1 - x|}, (2 - x)^3, \sqrt{|3 - x|}\} \\ &\textit{subject to} \quad 1 \leq x \leq 3. \end{aligned}$$

For the DC function f, we have $f = g - h$, where g, h are convex functions given by

$$\begin{aligned} g(x) &= 6x^2 - 12x + 8 + \max\{0, -x^3\} - \log(x) := G(x) - \log(x), \\ h(x) &= \max\{-\sqrt{|3 - x|} + G(x), -\sqrt{|1 - x|} + G(x), \max\{0, x^3\}\}. \end{aligned}$$

The problem attains an optimal solution at $x^* = 3$ with minimum value $-1 - \log 3$.

Example 6.1.3. *Consider the example*

$$\begin{aligned} &\textit{minimize} \quad f(x_1, x_2) := 0.03(x_1^2 + x_2^2) - \cos(x_1)\cos(x_2) \\ &\textit{subject to} \quad -6 \leq x_1 \leq 4, \\ &\qquad\qquad\quad -5 \leq x_2 \leq 2. \end{aligned}$$

The convex function $k(x_1^2 + x_2^2)$, $k > 0$ can be used to obtain different DC representations of the objective function f. We consider $k = 1$ only. For the DC function f, we have $f = g - h$, where g, h are convex functions given by

$$g(x) = 1.03(x_1^2 + x_2^2) - \cos(x_1)\cos(x_2), \quad h(x) = (x_1^2 + x_2^2).$$

The problem attains an optimal solution at $(0, 0)$ with minimum value -1.

Example 6.1.4. *Let $c_1 = 0.09$ and $c_2 = 0.1$. Consider the example*

$$\begin{aligned} \textit{minimize} \quad & f(x_1, x_2) := (x_1^2 + c_1 x_1)(x_2^2 + c_2 x_2) \\ \textit{subject to} \quad & -2 \leq x_1 \leq 1, \\ & -2 \leq x_2 \leq 1. \end{aligned}$$

Using the convex function $k(x_1^2 + x_2^2)$, $k > 0$, one can obtain different DC representations of f. We consider $k = 7.5$ and 8.5; solve the same example for each choice of k separately. For the DC function f, we have $f = g - h$, where g, h are convex functions given by

$$g(x) = (x_1^2 + 0.09x_1)(x_2^2 + 0.1x_2) + k(x_1^2 + x_2^2), \quad h(x) = k(x_1^2 + x_2^2).$$

Example 6.1.5. *Consider the example*

$$\begin{aligned} \textit{minimize} \quad & f(x_1, x_2) := \frac{1}{4}(x_1 + x_2)^2 - \frac{1}{4}(x_1 - x_2)^2 \\ \textit{subject to} \quad & -2 \leq x_1 \leq 3, \\ & -3 \leq x_2 \leq 4. \end{aligned}$$

For the DC function f, we have $f = g - h$, where g, h are convex functions given by

$$g(x) = \frac{1}{4}(x_1 + x_2)^2, \quad h(x) = \frac{1}{4}(x_1 - x_2)^2.$$

The problem attains an optimal solution at (3,-3) with minimum value -9.

Example 6.1.6. *Consider the example*

$$\begin{aligned} \textit{minimize} \quad & f(x_1, x_2) := -\sin(\sqrt{3x_1 + 2x_2 + |x_1 - x_2|}) \\ \textit{subject to} \quad & 0 \leq x_1 \leq 5, \\ & 0 \leq x_2 \leq 5. \end{aligned}$$

A DC representation of f can be obtained as follows: $f = g - h$, where g, h are convex functions given by

$$g(x) = 5(x_1^2 + x_2^2), \quad h(x) = \sin(\sqrt{3x_1 + 2x_2 + |x_1 - x_2|}) + 5(x_1^2 + x_2^2).$$

Example 6.1.7. *Consider the example for* $n \in \{1, 2, 3\}$ *and* $m \in \{2, 3\}$:

$$\begin{aligned} \textit{minimize} \quad & f(x) := -\sum_{i=1}^{m} \frac{1}{\|x - a^i e\|^2 + c_i} \\ \textit{subject to} \quad & 0 \le x_j \le 10, \quad j = 1, \ldots, n, x \in \mathbb{R}^n, \end{aligned}$$

where $e \in \mathbb{R}^n$ *is the vector of ones, and* $a^i \in \mathbb{R}_+$, $c_i \in \mathbb{R}_{++}$ *are parameters of the problem, see Table 6.1. Using the convex function* $k\|x\|^2$, $k > 0$, $x \in \mathbb{R}^n$, *one can obtain different DC representations of the objective function* f. *We consider* $k = 1$ *only. For the DC function* f, *we have* $f = g - h$, *where* g, h *are convex functions given by*

$$g(x) = f(x) + k\|x\|^2, \quad h(x) = k\|x\|^2.$$

Table 6.1: Parameters for Example 6.1.7

i	1	2	3
c_i	0.70	0.73	0.76
a^i	4.0	2.5	7.5

Example 6.1.8. *Consider the example*

$$\begin{aligned} \textit{minimize} \quad & f(x) := \sum_{i=1}^{3} |2 - x_i|^3 \\ \textit{subject to} \quad & 1 \le x_i \le 3, \quad i = 1, 2, 3. \end{aligned}$$

The objective function f *is a convex function on the feasible region. We still form a DC decomposition* $f = g - h$ *[2, Problem 10.4], where* g, h *are convex functions given by*

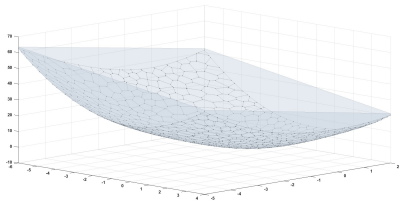
$$g(x) = \sum_{i=1}^{3} |2 - x_i|^3 + \|x\|^2, \quad h(x) = \|x\|^2.$$

The problem attains an optimal solution at (2,2,2) with minimum value 0.

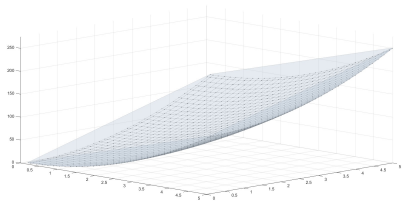
These examples are solve with Algorithms 1, 2, 3, and the algorithms from the literature (PS and DCECAM). Epigraphs of ϵ-polyhedral approximations of g returned by Algorithm 1 for Examples 6.1.3, 6.1.6, and 6.1.7 are shown in Figure 6.1. Also, outer-approximations of

epi g for the same examples, outputted by Algorithm 3 upon termination, are shown in Figure 6.2. In Section 6.2, we compare all five algorithms for the univariate examples, whereas in Section 6.3, we compare Algorithms 1, 2, and 3 with DCECAM for bivariate and trivariate examples.

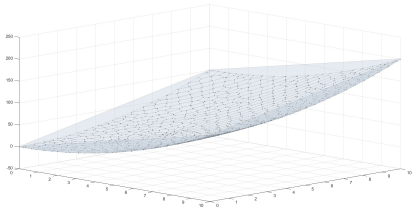
Tables 6.2-6.10 present the computational results, which show the algorithm (Alg), the stopping condition (ϵ), the solution returned by the algorithms (x^ϵ), the number of iterations, the algorithm's run time in seconds (t), the number of times the vertex enumeration function is called (vert_enum), the number of LPs solved (LP), the number of times the function g is called (g_eval), and the objective function value at x^ϵ (z^ϵ). Note that for Algorithms 1, 2, and 3, ϵ is also the approximation error, that is x^ϵ is an ϵ-solution. However, for DCECAM and PS this is not guaranteed in general, see Remark A.2.3.



(a) Example 6.1.3 with $\epsilon = 0.1$



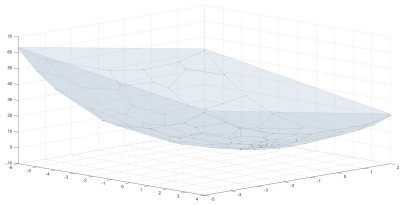
(b) Example 6.1.6 with $\epsilon = 0.1$



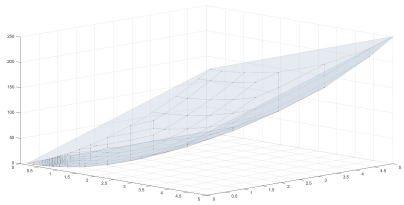
(c) Example 6.1.7 ($m = n = 2$) with $\epsilon = 0.1$

Figure 6.1: Epigraphs of ϵ-polyhedral approximations of g obtained from Algorithm 1 for some examples

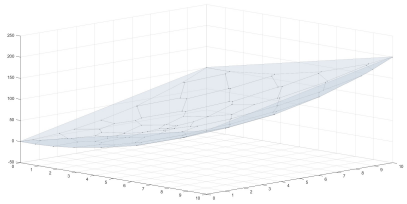
For each example, we set a time limit of one hour. Examples 6.1.3, 6.1.4, and 6.1.6 for $\epsilon = 0.01$ cannot be solved by Algorithm 1 within the time limit. In the case of DCECAM, for examples with two variables, Examples 6.1.4-6.1.6, we set a time limit of three hours. Examples 6.1.4 and 6.1.5 for $\epsilon = 0.01$ and Example 6.1.6 for $\epsilon = 0.1$ and 0.01 did not terminated within the time limit.



(a) Example 6.1.3 with $\epsilon = 0.01$



(b) Example 6.1.6 with $\epsilon = 0.01$



(c) Example 6.1.7 ($m = n = 2$) with $\epsilon = 0.1$

Figure 6.2: Polyhedral approximations of epi g obtained from Algorithm 3 upon termination

6.2 Comparison of Algorithms for $n = 1$

In this section, we evaluate the performances of Algorithms 1, 2 and 3 with DCECAM and Pijavski-Shubert (PS) algorithms, for univariate examples, in particular with Examples 6.1.1, 6.1.2 and 6.1.7 ($n = 1$), see Tables 6.2-6.4. Tables 6.2-6.4 show that, regardless of the algorithm, the iterations, run time, vertex enumeration call, number of LPs, and g_eval increase when a smaller value of ϵ is used. According to these tables, both PS and DCECAM take a greater amount of time to run than Algorithms 1, 2, and 3. Moreover, the PS method's run time is more than the run time for DCECAM. Furthermore, one can observe that the objective function value returned by DCECAM and PS are not close to the z^ϵ values returned by the Algorithms 1, 2, and 3. Also, as stated in Remark A.2.3, they are not ϵ-optimal solutions.

The x^ϵ and z^ϵ values returned by Algorithms 1, 2, and 3 are the same. Moreover, the run times of these three algorithms are almost the same for most instances. They return the solution within a few seconds. Algorithm 1 runs relatively faster than Algorithm 2 in some instances; see Tables 6.3 and 6.4. However, we observe that in Example 6.1.7, for $\epsilon = 0.01$ and 0.001, the run times of Algorithms 1 and 2 increase and are greater than Algorithm 3, see Table 6.4. In terms of time, Algorithm 3 outperforms both Algorithms 1 and 2.

Table 6.2: Numerical results for Example 6.1.1. For this example, the optimal objective value is 1 when $a \in \{1.5, 9\}$ and 0 when $a = 0$.

a	Alg	ϵ	x^ϵ	iterations	t (sec)	vert_enum	LP	g_eval	z^ϵ
9	Alg 1	0.1	1	4	0.4698	5	-	26	1
	Alg 2		1	2	0.3852	3	-	11	1
	Alg 3		1	1	0.1928	2	-	3	1
	PS		1.1766	9	1.4213	-	19	48	4.3404
	DCECAM		1.9963	22	1.5572	-	-	-	8.9833
	Alg 1	0.01	1	16	1.2139	17	-	188	1
	Alg 2		1	4	1.0333	5	-	35	1
	Alg 3		1	1	0.1595	2	-	3	1
	PS		1.0577	32	4.4899	-	65	163	2.9986
	DCECAM		1.9559	140	9.6758	-	-	-	8.7994
1.5	Alg 1	0.1	1	4	0.3753	5	-	26	1
	Alg 2		1	2	0.4095	3	-	11	1
	Alg 3		1	2	0.2222	3	-	4	1
	PS		1.3723	15	2.2056	-	31	78	1.1623
	DCECAM		1.9963	22	1.5366	-	-	-	1.4972
	Alg 1	0.01	1	16	1.2174	17	-	188	1
	Alg 2		1	4	1.0335	5	-	35	1
	Alg 3		1	2	0.2232	3	-	4	1
	PS		1.354	8.3606	60	-	121	303	1.1621
	DCECAM		1.9559	140	9.587	-	-	-	1.4666
0	Alg 1	0.1	2.444	4	0.4089	5	-	26	0.08753
	Alg 2		2.444	2	0.3538	3	-	11	0.08753
	Alg 3		2.444	3	0.2823	4	-	5	0.08753
	PS		1.9404	23	3.3058	-	47	118	2.117×10^{-4}
	DCECAM		1.9963	22	1.5842	-	-	-	5.0653×10^{-8}
	Alg 1	0.01	1.8025	16	1.0629	17	-	188	0.007704
	Alg 2		2.1975	4	1.2333	5	-	35	0.007704
	Alg 3		2.1975	5	0.3972	6	-	7	0.007704
	PS		1.9926	115	16.017	-	231	578	4.052×10^{-7}
	DCECAM		1.9559	140	9.6372	-	-	-	8.5766×10^{-5}

Table 6.3: Numerical results for Example 6.1.2. For this example, the optimal objective value is $-1 - \log 3$.

Alg	ϵ	x^ϵ	iterations	t (sec)	vert_enum	LP	g_eval	z^ϵ
Alg 1	0.1	3	8	0.6682	9	-	64	-2.099
Alg 2		3	3	0.7495	4	-	19	-2.099
Alg 3		3	3	0.4002	4	-	5	-2.099
PS		2.8975	63	14.833	-	127	318	-1.787
DCECAM		1.0762	46	10.9946	-	-	-	0.2025
Alg 1	0.01	3	32	2.1754	33	-	628	-2.099
Alg 2		3	5	2.3752	6	-	67	-2.099
Alg 3		3	3	0.4007	4	-	5	-2.099
PS		2.9785	183	57.4563	-	367	918	-2.028
DCECAM		1.058	219	45.2085	-	-	-	0.1844

Table 6.4: Numerical results for Example 6.1.7 when $n = 1$.

m	Alg	ϵ	x^ϵ	iterations	t (sec)	vert_enum	LP	g_eval	z^ϵ
3	Alg 1	1	3.3319	8	0.7981	9	-	64	-1.6307
	Alg 2		3.3319	3	0.907	4	-	19	-1.6307
	Alg 3		3.3319	7	0.6294	8	-	9	-1.6307
	PS		4.0184	74	13.3359	-	149	373	-1.8349
	DCECAM		0.0012	60	5.6345	-	-	-	-0.2209
	Alg 1	0.1	4.0397	22	2.0942	23	-	323	-1.8264
	Alg 2		4.0397	5	1.8484	6	-	7	-1.8264
	Alg 3		4.0397	11	0.9006	12	-	13	-1.8264
	PS		3.9283	237	39.3572	-	475	1188	-1.8531
	DCECAM		0.0650	182	16.9819	-	-	-	-0.2298
	Alg 1	0.01	3.9126	64	5.1474	65	-	2276	-1.8534
	Alg 2		3.9126	6	5.6964	7	-	131	-1.8534
	Alg 3		3.9126	13	1.0037	14	-	15	-1.8534
	PS		3.9192	517	85.6214	-	1035	2588	-1.8534
	DCECAM		0.0728	508	46.7022	-	-	-	-0.2309
	Alg 1	0.001	3.9235	239	22.2418	240	-	29401	-1.8533
	Alg 2		3.9235	8	20.3759	9	-	481	-1.8533
	Alg 3		3.9235	16	1.1947	17	-	18	-1.8533
	PS		3.9125	1252	206.0144	-	2505	6263	-1.8534
	DCECAM		0.0732	2024	188.4087	-	-	-	-0.2310
2	Alg 1	1	3.3686	8	0.7265	9	-	64	-1.5838
	Alg 2		3.3686	3	0.8302	4	-	19	-1.5838
	Alg 3		3.3686	6	0.5335	7	-	8	-1.5838
	PS		4.0131	71	11.045	-	143	358	-1.7594
	DCECAM		0.0048	60	5.1678	-	-	-	-0.2038
	Alg 1	0.1	4.0546	21	1.6242	22	-	298	-1.7403
	Alg 2		4.0546	5	1.8602	6	-	45	-1.7403
	Alg 3		4.0546	9	0.7117	10	-	11	-1.7403
	PS		3.9234	220	33.8892	-	441	1103	-1.7795
	DCECAM		0.0676	182	15.3291	-	-	-	-0.2123
	Alg 1	0.01	3.9287	64	4.7067	65	-	2276	-1.7791
	Alg 2		3.9287	6	5.1919	7	-	131	-1.7791
	Alg 3		3.9287	13	0.9441	14	-	15	-1.7791
	PS		3.9073	524	80.5228	-	1049	2623	-1.7802
	DCECAM		0.0696	504	42.1035	-	-	-	-0.2126
	Alg 1	0.001	3.8961	246	21.1839	247	-	31123	-1.7801
	Alg 2		3.8961	8	19.1861	9	-	495	-1.7801
	Alg 3		3.8961	16	1.1541	17	-	18	-1.7801
	PS		3.9031	1281	196.7629	-	2563	6408	-1.7802
	DCECAM		0.0703	2028	171.4137	-	-	-	-0.2127

6.3 Comparison of Algorithms for $n > 1$

In this section, we compare the performances of Algorithms 1, 2, and 3 among each other and with DCECAM [2] in the multivariate case. In particular, we compare the algorithms on three bivariate examples and a trivariate example. Tables 6.5-6.10 illustrate the results of Examples 6.1.3-6.1.8, respectively.

Table 6.5: Numerical results for Example 6.1.3. For this example, the optimal objective value is -1.

Alg	ϵ	x^ϵ	iterations	t (sec)	vert_enum	g_eval	z^ϵ
Alg 1	1	-0.3673, -0.3591	61	6.9231	62	4093	-0.8659
Alg 2		0.0339, -0.0323	4	16.7443	5	263	-0.999
Alg 3		0.0167, 0.1234	32	3.4446	33	34	-0.992
Alg 1	0.1	-0.0494, 0.0398	504	126.249	505	257046	-0.9979
Alg 2		-0.1409, 0.0085	7	136.7298	8	2122	-0.989
Alg 3		0.1004, -0.0532	53	5.6397	54	55	-0.993
Alg 1	0.01	-	-	-	-	-	-
Alg 2		0.0167, 0.0251	9	1338	10	19269	-0.9995
Alg 3		0.0107, -0.0448	64	6.9345	65	66	-0.999

Table 6.6: Numerical results for Example 6.1.4.

k	Alg	ϵ	x^ϵ	iterations	t (sec)	vert_enum	g_eval	z^ϵ
7.5	Alg 1	1	0.0725, 0.6398	62	7.2871	63	4222	0.0056
	Alg 2		-0.0828, -0.5298	4	18.8544	5	282	-0.0154
	Alg 3		0.8836, -0.1089	42	4.6612	43	44	8.34×10^{-4}
	DCECAM		0.0220, 0.0234	59	7.3986	-	-	7.1149×10^{-6}
	Alg 1	0.1	-0.1290, 0.4054	523	137.9944	524	276673	0.001
	Alg 2		-0.0012, -0.2566	7	154.7937	8	2294	-4.28×10^{-6}
	Alg 3		-1.7261, -0.0437	223	32.7173	224	225	-0.0069
	DCECAM		0.0032, 0.0017	2570	316.1815	-	-	5.1563×10^{-8}
	Alg 1	0.01	-	-	-	-	-	-
	Alg 2		-1.9316, -0.0319	9	1594	10	21391	-0.00773
	Alg 3		-1.9120, -0.0507	759	307.3269	760	761	-0.0087
	DCECAM		-	-	-	-	-	-
8.5	Alg 1	1	0.2702, -0.0160	69	8.2273	70	5181	-1.3081×10^{-4}
	Alg 2		0.2410, -0.1080	5	19.6505	6	294	6.89×10^{-5}
	Alg 3		0.0111, 0.2633	49	5.4364	50	51	1.074×10^{-4}
	DCECAM		0.0846, 0.0213	59	7.3122			3.8164×10^{-5}
	Alg 1	0.1	-1.7133, -0.0677	578	171.8382	579	337558	-0.0061
	Alg 2		-0.5922, -0.2318	7	184.4126	8	2742	-0.00909
	Alg 3		-0.0354, 0.1663	244	37.0946	245	246	-8.56×10^{-5}
	DCECAM		0.0115, -0.0067	2192	268.9987			-7.2966×10^{-7}
	Alg 1	0.01	-	-	-	-	-	-
	Alg 2		-1.9823, -0.0420	9	1856	10	24149	-0.00914
	Alg 3		-1.9481, -0.0527	801	343.8139	802	803	-0.009
	DCECAM		-	-	-	-	-	-

Table 6.7: Numerical results for Example 6.1.5. For this example, the optimal objective value is -9.

Alg	ϵ	x^ϵ	iterations	t (sec)	vert_enum	g_eval	z^ϵ
Alg 1	1	3,-3	4	0.6008	5	46	-9
Alg 2		3, -3	2	1.2704	3	19	-9
Alg 3		3, -3	Initial approx	0.172	1	2	-9
DCECAM		2.1052, -2.1233	242	25.5399	-	-	-4.47
Alg 1	0.1	3,-3	16	1.7787	17	358	-9
Alg 2		3, -3	3	2.6043	4	41	-9
Alg 3		3, -3	1	0.2839	2	3	-9
DCECAM		2.6111, -2.4060	54176	10468	-	-	-6.2823
Alg 1	0.01	3, -3	32	3.3925	33	1222	-9
Alg 2		3, -3	5	9.9531	6	161	-9
Alg 3		3, -3	1	0.2845	2	3	-9
DCECAM		-	-	-	-	-	-

Table 6.8: Numerical results for Example 6.1.6.

Alg	ϵ	x^ϵ	iterations	t (sec)	vert_enum	g_eval	z^ϵ
Alg 1	1	0.4062, 0.4063	80	7.8058	81	4594	-0.989
Alg 2		0.4062, 0.4063	6	9.8374	7	181	-0.989
Alg 3		0.4062, 0.4062	49	4.8789	50	51	-0.989
DCECAM		0.0063, 0.0039	5639	669.3728	-	-	-0.1698
Alg 1	0.1	0.4828, 0.4828	1088	641.5488	1089	719902	-0.989
Alg 2		0.4828, 0.4828	10	137.2688	11	2471	-0.989
Alg 3		0.4828, 0.4828	112	11.7933	113	114	-0.989
DCECAM		-	-	-	-	-	-
Alg 1	0.01	-	-	-	-	-	-
Alg 2		0.1, 0.7508	13	1612	14	19974	-1
Alg 3		0.3297, 0.5977	203	24.4363	204	205	-1
DCECAM		-	-	-	-	-	-

Table 6.9: Numerical results for Example 6.1.7 when $n > 1$.

n	m	Alg	ϵ	x^ϵ	iterations	t (sec)	vert_enum	g_eval	z^ϵ
2	3	Alg 1	1	3.9854, 4.2303	81	10.6402	82	7053	-1.5382
		Alg 2		4.2805,4.2805	5	36.6904	6	477	-1.3543
		Alg 3		2.1896, 2.7148	44	5.2056	45	46	-1.3431
		Alg 1	0.1	3.9982, 3.8967	688	264.7059	689	477478	-1.6486
		Alg 2		4.0076, 4.0076	8	274.2485	9	3527	-1.6576
		Alg 3		3.9836, 3.9006	64	7.7666	65	66	-1.6509
	2	Alg 1	1	4.1080, 3.7399	81	9.8562	82	7053	-1.4892
		Alg 2		3.6087, 3.6087	5	35.1613	6	487	-1.3074
		Alg 3		3.8252, 3.5221	41	4.7595	42	43	-1.326
		Alg 1	0.1	3.9807, 3.9538	663	235.2351	664	443553	-1.622
		Alg 2		4.0166, 4.0166	8	265.8499	9	3672	-1.6151
		Alg 3		3.9049, 4.0048	56	6.4291	57	58	-1.6116
3	3	Alg 1	10	3.9728, 2.1625, 5.3704	42	8.522	43	4859	-0.2784
		Alg 2		5.0467, 8.1733, 5.0467	3	61.1486	4	558	-0.1465
		Alg 3		5.1313, 2.0023, 4.6615	44	8.4153	45	46	-0.258
		Alg 1	5	4.8124, 3.4300, 3.5360	96	24.7274	97	24601	-0.6759
		Alg 2		2.8552, 2.2009, 4.9569	4	182.3466	5	1651	-0.3231
		Alg 3		6.5625, 8.3316, 8.0732	90	21.3977	91	92	-0.4117
		Alg 1	1	2.3415, 2.5233, 2.7010	873	3271.9	874	2181946	-1.4061
		Alg 2		7.3732, 7.3732, 7.7885	5	2774.2	6	23786	-1.1819
		Alg 3		3.7450, 4.2229, 4.0864	326	246.4885	327	328	-1.3719
	2	Alg 1	10	9.6078, 3.1195, 0	43	8.2798	44	5124	-0.0377
		Alg 2		1.3350, 1.3350, 2.4991	3	59.8155	4	554	-0.3486
		Alg 3		9.6078, 3.1195, 0	43	7.8921	44	45	-0.0377
		Alg 1	5	1.1767, 3.1310, 2.2516	96	24.89	97	24663	-0.4201
		Alg 2		2.8577, 2.2002, 4.9570	4	163.8912	5	1595	-0.3055
		Alg 3		1.1103, 3.1311, 2.2504	93	22.9599	94	95	-0.398
		Alg 1	1	2.4062, 2.3521, 2.4467	879	3278.8	880	2215911	-1.4292
		Alg 2		2.5436, 2.5436, 2.5436	5	2637.7	6	24376	-1.5008
		Alg 3		3.8799, 3.9657, 3.7438	275	163.9177	276	277	-1.438

Table 6.10: Numerical results for Example 6.1.8.

Alg	ϵ	x^ϵ	iterations	t (sec)	vert_enum	g_eval	z^ϵ
Alg 1	1	2.4727, 1.4164, 2.0006	34	5.5425	35	3110	0.304
Alg 2		2, 1.4429, 2.3875	3	45.8239	4	549	0.231
Alg 3		1.9999, 2.4718, 2.5835	34	5.4084	35	36	0.304
DCECAM		1.4905, 2.2810, 1.2426	26	9.0275	-	-	0.5889
Alg 1	0.1	2.0966, 1.7539, 2.0332	837	3262.1	838	2008183	0.0158
Alg 2		2, 1.7762, 2	6	2167.2	7	22456	0.0118
Alg 3		2.2362, 2.0484, 2.1551	71	13.4994	72	73	0.017
DCECAM		2.2352, 1.9342, 1.6282	40	16.0112	-	-	0.0647

6.3.1 Comparison of Algorithms 1, 2 and 3

Recall from Section 6.2 that, for $n = 1$, Algorithm 1 performs much faster than Algorithm 2 in constructing an ϵ-polyhedral approximation for a convex function. However, as the dimension of the decision space increases and the value of ϵ decreases, the former takes a greater amount of time to return the solution compare to the latter. For example, in Examples 6.1.3, 6.1.4, 6.1.6, for $\epsilon = 0.01$, Algorithm 1 did not terminate within an hour, see Tables 6.5, 6.6, 6.8. On the contrary, Algorithm 2 outperformed Algorithm 1. This may be due to the fact that Algorithm 1 considers only a single vertex at each iteration, whereas Algorithm 2 considers multiple vertices. As the dimension increases, the number of vertices obtained, through vertex enumeration technique at each iteration, increases drastically. Hence, Algorithm 1 takes a considerably greater amount of time to return an ϵ-polyhedral approximation everywhere on the domain. Moreover, this also implies that the number of times Algorithm 1 calls the vertex enumeration function is much greater than that of Algorithm 2.

On the other hand, Algorithm 3 works the fastest amongst all three algorithms. Note that Algorithms 1 and 2 are designed to return an ϵ-polyhedral approximation of g to solve the DC problem, whereas Algorithm 3 is designed to solve DC programming problems. Indeed, to have a fair evaluation of all three algorithms, we modify Algorithms 1 and 2 in such a way that both also solve and return an ϵ-optimal solution to the DC program, using the method from [1], once the required polyhedral approximation of g has been obtained, for some $\epsilon > 0$. In terms of time, Algorithm 3 works fastest because it traverses through the domain locally in searching for the ϵ-optimal solution. As the dimension of the decision space increases and ϵ decreases, the run time difference between Algorithm 3 and each of Algorithms 1 and 2 increases rapidly; see Tables 6.5, 6.6, 6.8-6.10. This is expected because the polyhedral approximation of g obtained, once Algorithm 3 terminates, may not necessarily be an ϵ-polyhedral underestimator. Like Algorithm 1, Algorithm 3 also considers only a single vertex at each iteration. Hence, the number of times the vertex enumeration function is called tends to be greater than that of Algorithm 2. Furthermore, as ϵ decreases, z^ϵ starts decreasing as it approaches the optimal value z^*. One can observe that, in most of the examples, the z^ϵ values returned by Algorithms 1, 2, and 3 are very close to each other. Instead, in some examples the z^ϵ values returned by all three algorithms are the same; see Tables 6.5, 6.7, 6.8. This implies that all three algorithms are consistent in providing an ϵ-optimal value to a DC programming problem.

6.3.2 Comparison of Algorithms 1, 2, and 3 with DCECAM

We observe from Tables 6.2-6.10 that as the dimension of the decision space increases, the run time for DCECAM increases. Indeed, DCECAM takes greater amount of time than each one of the Algorithms 1, 2, and 3 for each example. An important observation regarding the run time of DCECAM is that, for Example 6.1.8 with $\epsilon = 0.1$, the run time is low compared to the run times of some bivariate examples 6.1.4, 6.1.5; see Tables 6.6, 6.7, 6.10. This implies that the run time of DCECAM, in addition to the dimension of the decision space, may also depend on the structure and complexity of the problem. Another evidence to support this claim can be seen in Table 6.7. For $\epsilon = 0.1$, Algorithms 1, 2, and 3 solve Example 6.1.5 within a few seconds. Whereas, DCECAM takes nearly three hours to terminate for the same instance.

We also observe that the number of iterations for DCECAM increases with n. Furthermore, in most examples, the optimal values returned by DCECAM are in ϵ-distance to z^* even though this is not guaranteed by DCECAM; see also Remark A.2.3. There are some outliers where the difference between the optimal value returned by DCECAM and the original optimal value z^* is greater than ϵ, for some $\epsilon > 0$; see Tables 6.7, 6.8, 6.10.

Chapter 7

Conclusion

In this thesis, we consider DC programming problems and propose algorithms. First, we propose algorithms for approximating a convex function over a a feasible region X by a polyhedral minorant. The polyhedral underestimator of the first convex component of the DC function g is used to solve a DC programming problem approximately. The first algorithm proceeds by intersecting a halfspace to the epigraph of g at each iteration. Moreover, we have also proved that it works correctly. Algorithm 1 outputs an H-sequence of polytopes $(A^k)_{k\geq 0}$ to a compact set $A \subseteq \mathcal{C}$. Then, using results from [33, 34], we establish the convergence rate of Algorithm 1. We also propose a modification, namely Algorithm 2. In Algorithm 2, possibly multiple halfspaces are intersected to the epigraph of g, iteratively. Algorithm 2 is expected to work faster and provides a stronger polyhedral approximation than its predecessor, at each iteration. We prove the correctness and finiteness of Algorithm 2.

Moreover, we propose another algorithm (Algorithm 3) for solving DC programming problems over a compact set X. At each iteration, it keeps updating the polyhedral underestimator of g locally while searching for an ϵ-solution of the DC programming problem directly. We prove the correctness and finiteness of Algorithm 3. Moreover, we also show that the sequence $\{x^k\}_{k\geq 0}$, outputted by Algorithm 3, converges to the global minimizer of the DC programming problem.

As a future research direction, the convergence rate of Algorithm 2 could be established using similar means used in the case of Algorithm 1. However, as per the definition of H-sequence of outer approximating polytopes, only a single halfspace is added at each iteration.

Hence, the results from [33] cannot be applied directly here.

Furthermore, Löhne and Wagner proposed a method for solving polyhedral DC programs when the second component of the DC function h is polyhedral convex. In this case, one needs to use the duality theory introduced by Toland [47] and Singer [48], see [1, Section 5]. Another future research direction could be to work on the implementation of our proposed algorithms by approximating the conjugate of the second convex component of the DC function h^*.

Bibliography

[1] A. Löhne and A. Wagner, "Solving DC programs with a polyhedral component utilizing a multiple objective linear programming solver," *Journal of Global Optimization*, vol. 69, no. 2, pp. 369–385, 2017.

[2] A. Ferrer, A. Bagirov, and G. Beliakov, "Solving DC programs using the cutting angle method," *Journal of Global Optimization*, vol. 61, no. 1, pp. 71–89, 2015.

[3] S. Piyavskii, "An algorithm for finding the absolute extremum of a function," *Computational Mathematics and Mathematical Physics*, vol. 12, no. 4, pp. 57–67, 1972.

[4] B. O. Shubert, "A sequential method seeking the global maximum of a function," *SIAM Journal on Numerical Analysis*, vol. 9, no. 3, pp. 379–388, 1972.

[5] P. D. Tao and L. T. H. An, "Convex analysis approach to DC programming: theory, algorithms and applications," *Acta Mathematica Vietnamica*, vol. 22, no. 1, pp. 289–355, 1997.

[6] L. T. H. An and P. D. Tao, "DC programming and DCA: thirty years of developments," *Mathematical Programming*, vol. 169, no. 1, pp. 5–68, 2018.

[7] K. Holmberg and H. Tuy, "A production-transportation problem with stochastic demand and concave production costs," *Mathematical Programming*, vol. 85, no. 1, pp. 157–179, 1999.

[8] P. C. Chen, P. Hansen, B. Jaumard, and H. Tuy, "Solution of the multisource Weber and conditional Weber problems by DC programming," *Operations Research*, vol. 46, no. 4, pp. 548–562, 1998.

[9] R. Horst and N. V. Thoai, "DC programming: overview," *Journal of Optimization Theory and Applications*, vol. 103, no. 1, pp. 1–43, 1999.

[10] L. T. H. An and P. D. Tao, "Open issues and recent advances in DC programming and DCA," *Journal of Global Optimization*, pp. 1–58, 2023.

[11] L. T. H. An and P. D. Tao, "Large-scale molecular optimization from distance matrices by a DC optimization approach," *SIAM Journal on Optimization*, vol. 14, no. 1, pp. 77–114, 2003.

[12] P. D. Tao and L. T. H. An, "A DC optimization algorithm for solving the trust-region subproblem," *SIAM Journal on Optimization*, vol. 8, no. 2, pp. 476–505, 1998.

[13] L. T. H. An, P. D. Tao, and L. D. Muu, "A combined DC optimization ellipsoidal branch-and-bound algorithm for solving nonconvex quadratic programming problems," *Journal of Combinatorial Optimization*, vol. 2, pp. 9–28, 1998.

[14] M. Thiao, P. D. Tao, and L. T. H. An, "DC programming approach for a class of nonconvex programs involving l_0 norm," in *Modelling, Computation and Optimization in Information Systems and Management Sciences* (L. T. H. An, P. D. Tao, and P. Bouvry, eds.), pp. 348–357, Springer, 2008.

[15] H. Tuy, "Concave programming with linear constraints," *Proceedings of the USSR Academy of Sciences*, vol. 159, no. 1, pp. 32–35, 1964.

[16] P. D. Tao and L. T. H. An, "Difference of Convex functions optimization algorithms (DCA) for globally minimizing nonconvex quadratic forms on Euclidean balls and spheres," *Operations Research Letters*, vol. 19, no. 5, pp. 207–216, 1996.

[17] L. T. H. An, M. T. Belghiti, and P. D. Tao, "A new efficient algorithm based on DC programming and DCA for clustering," *Journal of Global Optimization*, vol. 37, no. 4, pp. 593–608, 2006.

[18] P. D. Tao, L. T. H. An, and L. H. Minh, "Optimization based DC programming and DCA for hierarchical clustering," *European Journal of Operational Research*, vol. 183, no. 3, pp. 1067–1085, 2006.

[19] P. D. Tao, L. T. H. An, and F. Akoa, "Combining DCA (DC algorithms) and interior point techniques for large-scale nonconvex quadratic programming," *Optimization Methods & Software*, vol. 23, no. 4, pp. 609–629, 2008.

[20] S. Bouallagui, P. D. Tao, and L. T. H. An, "Design of highly nonlinear balanced boolean functions using an hybridation of DCA and simulated annealing algorithm," in *Modelling, Computation and Optimization in Information Systems and Management Sciences* (L. T. H. An, P. D. Tao, and P. Bouvry, eds.), pp. 579–588, Springer, 2008.

[21] H. Konno, P. T. Thach, and H. Tuy, *Optimization on Low Rank Nonconvex Structures*, vol. 15. Springer, 2013.

[22] P. D. Tao, "Convergence of a subgradient method for computing the bound norm of matrices," *Linear Algebra and its Applications*, vol. 62, pp. 163–182, 1984.

[23] J. F. Toland, "On subdifferential calculus and duality in nonconvex optimization," *Bulletin de la Société Mathématique de France*, vol. 60, pp. 177–183, 1979.

[24] R. Horst and H. Tuy, *Global Optimization: Deterministic Approaches*. Springer, 1990.

[25] A. Strekalovsky and I. Tsevendorj, "Testing the R-strategy for a reverse convex problem," *Journal of Global Optimization*, vol. 13, no. 1, pp. 61–74, 1998.

[26] L. T. H. An and P. D. Tao, "The DC (Difference of Convex functions) programming and DCA revisited with DC models of real world nonconvex optimization problems," *Annals of Operations Research*, vol. 133, no. 1, pp. 23–46, 2005.

[27] H. P. Benson and R. Horst, "A branch and bound-outer approximation algorithm for concave minimization over a convex set," *Computers & Mathematics with Applications*, vol. 21, no. 6-7, pp. 67–76, 1991.

[28] H. P. Benson, "An outcome space branch and bound-outer approximation algorithm for convex multiplicative programming," *Journal of Global Optimization*, vol. 15, no. 4, pp. 315–342, 1999.

[29] W. Melo, M. Fampa, and F. Raupp, "Integrating nonlinear branch-and-bound and outer approximation for convex mixed integer nonlinear programming," *Journal of Global Optimization*, vol. 60, no. 2, pp. 373–389, 2014.

[30] S. vom Dahl and A. Löhne, "Solving polyhedral DC optimization problems via concave minimization," *Journal of Global Optimization*, vol. 78, no. 1, pp. 37–47, 2020.

[31] G. Beliakov, "A review of applications of the cutting angle methods," *Continuous Optimization*, pp. 209–248, 2005.

[32] Ç. Ararat, F. Ulus, and M. Umer, "Convergence analysis of a norm minimization-based convex vector optimization algorithm," *arXiv preprint arXiv:2302.08723*, 2023.

[33] A. V. Lotov, V. A. Bushenkov, and G. K. Kamenev, *Interactive Decision Maps: Approximation and Visualization of Pareto Frontier*, vol. 89. Springer, 2004.

[34] G. K. Kamenev, "A class of adaptive algorithms for approximating convex bodies by polyhedra," *Computational Mathematics and Mathematical Physics*, vol. 32, no. 1, pp. 136–152, 1992.

[35] Ç. Ararat, F. Ulus, and M. Umer, "A norm minimization-based convex vector optimization algorithm," *Journal of Optimization Theory and Applications*, pp. 1–32, 2022.

[36] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge university press, 2004.

[37] M. O'Searcoid, *Metric Spaces*. Springer, 2007.

[38] I. N. Keskin and F. Ulus, "Outer approximation algorithms for convex vector optimization problems," *Optimization Methods and Software*, pp. 1–33, 2023.

[39] R. T. Rockafellar, *Convex Analysis*, vol. 11. Princeton University Press, 1997.

[40] G. K. Kamenev and A. I. Pospelov, "Polyhedral approximation of convex compact bodies by filling methods," *Computational Mathematics and Mathematical Physics*, vol. 52, no. 5, p. 680, 2012.

[41] J. Jost, *Postmodern Analysis*. Springer, 2005.

[42] D. Charalambos and B. Aliprantis, *Infinite Dimensional Analysis: A Hitchhiker's Guide*. Springer-Verlag Berlin and Heidelberg GmbH & Company KG, 2006.

[43] P. Hansen and B. Jaumard, "Lipschitz optimization," in *Handbook of Global Optimization* (R. Horst and P. Pardalos, eds.), pp. 407–493, Springer, 1995.

[44] A. Löhne and B. Weißing, "The vector linear program solver bensolve–notes on theoretical background," *European Journal of Operational Research*, vol. 260, no. 3, pp. 807–813, 2017.

[45] H. Tuy, *Convex Analysis and Global Optimization*. Springer, 1998.

[46] R. Horst, P. M. Pardalos, and N. Van Thoai, *Introduction to Global Optimization*. Springer, 2000.

[47] J. F. Toland, "Duality in nonconvex optimization," *Journal of Mathematical Analysis and Applications*, vol. 66, no. 2, pp. 399–415, 1978.

[48] I. Singer, "A Fenchel-Rockafellar type duality theorem for maximization," *Bulletin of the Australian Mathematical Society*, vol. 20, no. 2, pp. 193–198, 1979.

[49] G. Beliakov, "Geometry and combinatorics of the cutting angle method," *Optimization*, vol. 52, no. 4-5, pp. 379–394, 2003.

[50] J. E. Kelley, Jr, "The cutting-plane method for solving convex programs," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 4, pp. 703–712, 1960.

[51] E. W. Cheney and A. A. Goldstein, "Newton's method for convex programming Tchebycheff and approximation," *Numerische Mathematik*, vol. 1, no. 1, pp. 253–268, 1959.

[52] L. M. Batten and G. Beliakov, "Fast algorithm for the cutting angle method of global optimization," *Journal of Global Optimization*, vol. 24, no. 2, p. 149, 2002.

Appendix A

A short description of some methods from the literature

In this chapter, we recall some methods from the literature to solve DC programs. Appendix A.1 recalls the method to compute the optimal solution of a DC program, by Löhne and Wagner [1], in the case when at least one of the components of the DC function is a polyhedral convex function. Appendix A.2 recalls two existing methods, from the literature, to solve DC programs. For one-dimensional DC programs, the Pijavski-Shubert Method [3, 4, 43] solves DC programs by solving LPs at each iteration. For multi-dimensional problems, we will look at the Difference of Convex Estimated Cutting Angle Method (DCECAM), a hybrid approach proposed by Bagirov, Beliakov and Ferrer [2], to solve DC programs using outer approximation and branch and bound techniques.

A.1 Löhne-Wagner Method (LWM) [1]

Consider the DC program

$$\min(g(x) - h(x)) \quad \text{subject to} \quad x \in \operatorname{dom} g, \tag{P_{LW}}$$

where both g and h are convex functions and at least one of them is a polyhedral convex function. The Löhne-Wagner Method (LWM) computes an exact optimal solution to the

(P_{LW}). It is based on enumerating the vertices of epi g, in case g is polyhedral, and of epi h^* in case h is polyhedral, where h^* is the convex conjugate of h. Here, we only explain the method for the case when g is polyhedral. In this case, (P_{LW}) can be expressed equivalently as

$$\min(r - h(x)) \quad \text{subject to} \quad (x, r) \in \operatorname{bd} \operatorname{epi} g.$$

Let $\{F_i \mid i = 1, \ldots, k\}$ be the finite set of all the facets of epi g. Then $\operatorname{bd} \operatorname{epi} g = \bigcup_{i=1,\ldots,k} F_i$. Moreover, following two results hold [1].

1) For $i = 1, \ldots, k$, let (x^i, r^i) be an optimal solution of

$$\min(r - h(x)) \quad \text{subject to} \quad (x, r) \in F_i. \tag{P_i}$$

Let $j \in \arg\min\{r^i - h(x^i) \mid i = 1, \ldots, k\}$. Then, (x^j, r^j) is an optimal solution of (P_{LW}). Problem (P_{LW}) has an optimal solution if and only if, for all $i \in \{1, \ldots, k\}$, (P_i) has an optimal solution [1, Corollary 10].

2) Assume that (P_{LW}) has an optimal solution. Every optimal solution $(\bar{x}, \bar{r})$ of

$$\min(r - h(x)) \quad \text{subject to} \quad (x, r) \in \operatorname{vert} \operatorname{epi} g$$

is also an optimal solution of (P_{LW}) [1, Corollary 11].

The Löhne-Wagner Method is based on these observations and can be seen in Algorithm 4. The correctness of the method is shown in [1, Corollary 14].

Algorithm 4 Löhne-Wagner Method (g is polyhedral).

1: **Input:** Problem (P_{LW}), g is polyhedral, epi g has a vertex.
2: Compute $\mathcal{V} \coloneqq \operatorname{vert} \operatorname{epi} g$ using *bensolve tools.*
3: Let $(x^j, r^j) \in \arg\min_{(x,r)\in\mathcal{V}}(r - h(x))$.
4: Then $r^j - h(x^j) = \min_{(x,r)\in\mathcal{V}}(r - h(x))$.
5: **return** (x^j, r^j).

A.2 Cutting Angle Methods

Consider the DC problem as given in (P).

In this section, we recall two algorithms from the literature that solve DC programs. The Pijavski-Shubert (PS) Method [3, 4, 43] solves DC programming problems by a cutting plane method which includes solving LPs at each iteration. However, this method becomes computationally inefficient for higher dimensions as the number of LPs, at each iteration, increases exponentially.

In 2003, Beliakov [49] proposed that the Estimated Cutting Angle Method (ECAM), designed by Kelley [50] and Cheney and Goldstein [51] for convex programs, could be implemented on Lipschitz objective functions. The details of these can be viewed in [31, 49]. Later, in 2015, Beliakov, Bagirov and Ferrer [2] proposed another algorithm, DCECAM, to solve DC programming problems based on ECAM.

Both PS and DCECAM require that the first component of the DC objective function, in addition to being convex, has to be a Lipschitz continuous function on the feasible set. Both methods are applied to the first component of the DC function g. Starting with an initial underestimate H^0 of g, at each iteration k, the support functions are appended to the preceding underestimate to establish an increasing sequence of underestimates H^k of g.

At iteration k, the support function is

$$h_k(x) \coloneqq \min_{i=1,\ldots,n+1} (g(x^k) - L(x_i^k - x_i)),$$

where L is the Lipschitz constant of g and x^k is the global minimum of H^{k-1}. Then, the underestimate H^k is constructed as

$$H^k(x) = \max\{H^{k-1}(x), h_k(x)\} = \max_{0\leq j\leq k} h_j(x). \tag{A.2.1}$$

Once the underestimate is updated, the next step is to compute all local minima of the underestimate function. Next, PS method and DCECAM will be shortly detailed, respectively, in Sections A.2.1 and A.2.2.

A.2.1 The Pijavski-Shubert Method [3, 4]

In this section, we explain the PS method as in [31]. In case of one-dimensional problems, the support functions can be taken as

$$h_k(x) = g(x^k) - L|x^k - x|. \tag{A.2.2}$$

Then the problem of enumerating the local minima of the underestimate H^k can be written as

$$\min_{x \in X} H^k(x) = \min_{x \in X} \max_{0 \leq j \leq k} \{g(x^k) - L|x^k - x|\}. \tag{A.2.3}$$

While implementing the PS method, we adopted an approach where x^k values are sorted in the ascending order, $\tilde{x}^k$. To do that, the domain X is divided into $k+1$ sub-domains. Let the sub-domains be $[l, \tilde{x}^1], [\tilde{x}^1, \tilde{x}^2], \ldots, [\tilde{x}^i, \tilde{x}^{i+1}], \ldots, [\tilde{x}^k, u]$. Linearizing (A.2.3) gives us $k+1$ LP programs, with different sub-domains. The $(j+1)^{st}$ LP program, where we set $\tilde{x}^0 = l$, $\tilde{x}^{k+1} = u$, is as follows

$$\begin{aligned} \text{minimize} \quad & \xi \\ \text{subject to} \quad & \xi \geq g(\tilde{x}^j) - L(x - \tilde{x}^j), j = 0, \ldots, i, \\ & \xi \geq g(\tilde{x}^j) - L(\tilde{x}^j - x), j = i+1, \ldots, k, \\ & \tilde{x}^j \leq x \leq \tilde{x}^{j+1}. \end{aligned} \tag{A.2.4}$$

At iteration k, $k+1$ LP models (A.2.4) are solved for $j \in \{0, \ldots, k\}$. Each LP yields a local minimum of the original problem. Let Z^* be the set of local minima and let $x^* \in Z^*$ be the local minimum that yields the smallest value of the function $H^k(x) - h(x)$. Then x^* is the solution at iteration k. The PS method terminates when $|g(x^*) - H^k(x^*)| \leq \epsilon$ for some predetermined $\epsilon > 0$. Otherwise, the underestimate is updated using x^* and the algorithm continues to iterate. It is shown that the algorithm converges to a global minimizer when ϵ is set to zero [3, Section 1], [4, Section 2], [2, Section 6].

A.2.2 The DCECAM Algorithm [2]

The following two theorems, from the literature, provide useful results on the enumeration of local minima of the underestimate H^k when $X \subseteq \mathbb{R}^n$. In particular, assume that X is the unit

simplex in $\mathbb{R}^n$. Let $l^k \in \mathbb{R}^{n+1}$ be support vectors defined as $l_i^k \coloneqq \frac{g(x^k)}{L} - x_i^k,\ i = 1, \ldots, n+1$, at iteration k, where x^k is the global minimum of H^{k-1} over X and $x_{n+1}^k \coloneqq 1 - \sum_{i=1}^n x_i^k$.

Theorem A.2.1. *[2, Theorem 4.1] A necessary and sufficient condition for a point $x^* \in \operatorname{ri} X$ to be a local minimizer of H^k is that there exist an index set $J = \{k_1, k_2, \ldots, k_{n+1}\} \subseteq \{0, \ldots, k\}$ such that*

$$H^k(x^*) = L(l_1^{k_1} + x_1^*) = L(l_2^{k_2} + x_2^*) = \cdots = L(l_{n+1}^{k_{n+1}} + x_{n+1}^*),$$

and

$$(l_i^{k_i} + x_i^*) < (l_j^{k_i} + x_j^*), \quad \forall i, j \in \{1, \ldots, n+1\} \text{ with } j \neq i.$$

Let x^* be a local minimizer of the underestimate function H^k, which corresponds to some index set J satisfying conditions of Theorem A.2.1. The support vectors can be represented using a matrix whose rows are the support vectors l^{k_i}, that is,

$$M \coloneqq \begin{pmatrix} l_1^{k_1} & l_2^{k_1} & \ldots & l_{n+1}^{k_1} \\ l_1^{k_2} & l_2^{k_2} & \ldots & l_{n+1}^{k_2} \\ \vdots & \vdots & \ddots & \vdots \\ l_1^{k_{n+1}} & l_2^{k_{n+1}} & \ldots & l_{n+1}^{k_{n+1}} \end{pmatrix}.$$

Theorem A.2.2. *[2, Theorem 4.2] Let x^* denote a local minimizer of H^k in $\operatorname{ri} X$ and $d \coloneqq H^k(x^*)$. Then matrix M corresponding to x^* enjoys the following properties:*

(1) $\forall i, j \in \{1, \ldots, n+1\},\ i \neq j : l_i^{k_j} > l_i^{k_i}$;

(2) $\forall r \in \{0, \ldots, k\} \setminus J,\ \exists i \in \{1, \ldots, n+1\} : M_{ii} = l_i^{k_i} \geq l_i^r$;

(3) $d = \frac{L(Trace(M)+1)}{n+1}$;

(4) $x_i^* = \frac{d}{L} - l_i^{k_i},\ i = 1, \ldots, n+1.$

Condition (1) of Theorem A.2.2 states that the diagonal elements of matrix M are dominated by their respective columns, and condition (2) states that no support vector l^r (for $r \notin J$) strictly dominates the diagonal of M. The approach taken in [49, 52] is to enumerate all combinations of corresponding support vectors with properties (1)–(2), and compute the

local minima and their values by using (3)–(4). Then, x^{k+1} is defined such that

$$H^k(x^{k+1}) - h(x^{k+1}) = \min\{H^k(z) - h(z) \mid z \text{ is a local minimum of } H^k(x)\}.$$

The underestimate function $H^k(x)$ is updated to $H^{k+1}(x)$ by the support function corresponding to the point x^{k+1}. In [2, Section 6], it has been shown that the sequence of solutions returned by DCECAM converges to a global minimizer of the original DC problem. See Algorithms 5 and 6 for the details.

Remark A.2.3. *Even though both PS and DCECAM converge to global minimizers when ϵ is set to zero, they do not guarantee returning ϵ-solutions, as in Definition 2.2.2 if $\epsilon > 0$ in the algorithms.*

Algorithm 5 DCECAM Algorithm

T^K denotes the tree of local minima of functions $H^{n+1}, \ldots, H^K, V^K$ denotes the priority queue containing the leaves of the tree arranged in the increasing order by evaluating $H^K(z) - h(z)$ at every local minimum z of H^K.
Initialization: Take the initial points $x^k \in \mathbb{R}^{n+1}$, $k = 1, \ldots, n+1$, to be the extreme points of the simplex in $\mathbb{R}^n$ and construct the support vectors $l_i^k = \frac{g(x^k)}{L} - x_i^k$;
Set $K := n+1$, $M_{root} := \{l^1, l^2, \ldots, l^{n+1}\}$, $T^{n+1} = V^{n+1} := M_{root}$;
$f_{best} := \min_{k\in\{1,\ldots,n+1\}}(g(x^k) - h(x^k))$;
continue := true;
while (continue) **do**
    For every local minimum $M \in V^K$, compute the objective function value d using condition (3) of Theorem A.2.2;
    Choose M^*, corresponding to the support vectors that gives the global minimum of $H^K(x) - h(x)$, at the nodes of $H^K(x)$;
    Compute x^* from M^* using condition (4) of Theorem A.2.2;
    Evaluate $f^* := g(x^*) - h(x^*)$, $f_{best} := \min\{f_{best}, f^*\}$;
    Set $K \leftarrow K + 1$;
    Form l^K using $l_i^K := \frac{f(x^*)}{L} - x_i^*$;
    Update Tree (T^{K-1}, V^{K-1}, l^K); /*Algorithm 6*/
    if $K > K_{max}$ or $|f_{best} - (H^K(x^*) - h(x^*))| \leq \epsilon$ **then**
        continue $\leftarrow$ false;
    end if
end while
return (x^*, f^*)

Algorithm 6 Update Tree (T^{K-1}, V^{K-1}, l^K)

Inputs: The tree T^{K-1}, its leaves V^{K-1} and the new support vector l^K.
for all $M \in V^{K-1}$ **do**
  $V^{K-1} = V^{K-1} \setminus \{M\}$;
  Check M against condition (2) of Theorem A.2.2 with $l^r = l^K$, where $r \in \{0, \ldots, K-1\} \setminus J$;
  / Here J is the index set for support vectors corresponding to M./
  if condition (2) is satisfied **then**
    Add $n+1$ children to M;
    */each child node is a copy of M, with l^{k_i} replaced with l^K in the i^{th} child. */
    for each child node M_C of M **do**
      if condition (1) is not satisfied **then**
        delete this child node;
      else
        $V^{K-1} = V^{K-1} \cup \{M_C\}$.
      end if
    end for
  end if
end for
$V^K := V^{K-1}$, $T^K := T^{K-1} \cup V^K$.
if M is M_{root} **then**
  */we need to check this only once, at the first iteration */
  $T^K := T^{K-1}$;
  $V^K := leaves(T^K)$;
end if
return T^K, V^K

Appendix B

Codes

Here, you can find the codes for Algorithms 1, 2, and 3. The codes are implemented using MATLAB R2022a along with *bensolve tools* [44] to solve the vertex enumeration problems in each iteration, respectively.

B.1 Code for Algorithm 1

```
lb = ; %lower bounds
ub = ; %upper bounds
dimension = n;
epsilon = ; %epsilon values

[init_rep x_eps time vert_func_call first_func_call iterations
   vertices_count] = initial_approx(lb,ub,dimension,epsilon) %
   Outputs = time, vertices

if dimension < 3
    plot(init_rep);
else
    disp("Dimension too high; cannot plot the polyhedral
       function")
```

```
end

%This function evaluates function values at the x points
function [coordinate grad] = pconvex(dim, xval)
        syms x [1 dim]
        func = -(1/((norm(x-4))^2+0.7))-(1/((norm(x-2.5))
           ^2+0.73)) + (norm(x))^2;
        g = matlabFunction(func,'vars',{x});
        value = g(xval);
        coordinate = [];
        derivative = gradient(func);
        grad = [];
        for i = 1:dim;
            coordinate = [coordinate xval(i)];
            deriv_val = matlabFunction(derivative(i), 'vars',{
               x});
            grad = [grad deriv_val(xval)];
        end
        coordinate = [coordinate value];
        grad = [grad -1];
end

%This function gives the average of the bounds to find a
%non-vertical/non-horizontal hyperplane
function avg_hyp = another_hyperplane(lower_bound, upper_bound,
    dim)
        hrep = [];
        avg = [];
        for i = 1:dim;
            val = (lower_bound(i) + upper_bound(i))/2;
            avg = [avg val];
        end
        [avg_bound] = avg;
        [coordinate gradient] = pconvex(dim, avg_bound);
```

```
        avg_hyp = [hrep; gradient coordinate*gradient'];
end

%This function gives the initial representation (cylindrical
%representation) using box constraints
function [init_hrep] = initial_representation(lower_bound,
   upper_bound, dim, hyperplane_set)
        rep.B = [];
        rep.b = [];
        for i = 1:size(hyperplane_set,1)
            rep.B = [rep.B ; hyperplane_set(i,[1: dim+1])];
            rep.b = [rep.b ; hyperplane_set(i,dim+2)];
        end
        lower_bound = [lower_bound; -inf];
        upper_bound = [upper_bound; inf];
        rep.l = lower_bound;
        rep.u = upper_bound;
        init_hrep=polyh(rep,'h');
end

%This function checks whether a vertex satisfies the
   terminiating condition
function TorF = isavertex(v1,v2,epsilon)
        TorF=0;
        for i=1:size(v2,2)
            if norm(v1-v2(:,i)) < 10^-10
               TorF=1;
               break;
            end
        end
end

%This function seperates the vertices which violate the
   terminating
```

```
%condition from those who satisfy
function [unknown_vertices] = vertex_enumeration(representation
   ,known_vertices,epsilon)
        unknown_vertices=[];
        res=vrep(representation);
        vert=(res.V);
        for i=1:size(vert,2);
            if ~isavertex(vert(:,i),known_vertices,epsilon)
                unknown_vertices=[unknown_vertices vert(:,i)];
            end
        end
end

function [xstar] = LW(vert_epi,dim)

        syms x [1 dim]
        func = (norm(x))^2;%some other convex function h;
        h = matlabFunction(func,'vars',{x});
        minimal_set = [];
        for i = 1:size(vert_epi,2)
            vert_point = vert_epi(:,i);
            x_i = [];
            for j = 1:dim
                x_i = [x_i vert_point(j)];

                r = vert_point(dim+1);
            end
            val = r-h(x_i);
            minimal_set = [minimal_set val];
        end
        [min_val, min_index] = min(minimal_set);
        opt_soln = vert_epi(:,min_index);
        xstar = opt_soln(1:dim);
        %xstar = opt_soln(1:dim);
```

```
end

function [vstar, fcall epsval] = farthest_vertex(vertices,n)
        func_call = 0;
        syms x [1 n]
        func = -(1/((norm(x-4))^2+0.7))-(1/((norm(x-2.5))
           ^2+0.73))+ (norm(x))^2;%some other convex function
           h;
        g = matlabFunction(func,'vars',{x});
        vertex_set = [];
        for i = 1:size(vertices,2)
            vert_point = vertices(:,i);
            x_i = [];
            for j = 1:n
                x_i = [x_i vert_point(j)];
                r = vert_point(n+1);
            end
            val = g(x_i)-r;
            func_call = func_call +1;
            vertex_set = [vertex_set val];
        end
        [max_val, max_index] = max(vertex_set);
        vstar = vertices(:,max_index);
        fcall = func_call;
        epsval = max_val;
end

function [init_rep xstar time bensolve_number f1_eval k
   nvertices] = initial_approx(lower_bound, upper_bound, dim,
   epsilon)
        tic;
        hyperplanes = [];
        init_hyp = another_hyperplane(lower_bound, upper_bound
           , dim);
```

```
hyperplanes = [hyperplanes init_hyp];
init_rep = initial_representation(lower_bound,
   upper_bound, dim, hyperplanes);
v_known = [];
k = 0;
counter = 0;
f1_eval = 1;
vert_call = 0;
while counter>=0
    vertices = [];
    k = k + 1;
    vert = vertex_enumeration(init_rep,v_known,epsilon
      );
    vert_call = vert_call +1;
    for i = 1:size(vert,2)
       v = vert(:,i);
       vertices = horzcat(vertices,v);
    end
    [v fcall error] = farthest_vertex(vertices,dim);
    f1_eval = f1_eval + fcall;
    xdim = [];
    for i = 1:dim
        xdim = [xdim v(i)];
    end
    [x dx] = pconvex(dim,xdim);
    f1_eval = f1_eval + 1;
    v_known = [v_known v];
    if error <= epsilon
        final_v = v;
        k = k - 1;
        break
    else
        new_hyp = [];
        new_hyp = [new_hyp; dx x*dx'];
```

```
                hyperplanes = vertcat(hyperplanes, new_hyp);
                init_rep = initial_representation(lower_bound,
                   upper_bound,dim,hyperplanes);
            end
        end
        nvertices = size(vert,2);
        xstar = LW(vertices,dim);
        bensolve_number = vert_call;
        time = toc;
end
```

B.2 Code for Algorithm 2

```
lb = ; %lower bounds
ub = ; %upper bounds
dimension = n;
epsilon = ; %epsilon value

[init_rep final_vertices vertices_count x_eps iterations time
   vert_func_call first_func_call] = initial_approx(lb,ub,
   dimension,epsilon)

if dimension < 3
    plot(init_rep);
else
    disp("Dimension too high; cannot plot the polyhedral
       function")
end


%This function evaluates function values at the x points
function [coordinate grad] = pconvex(dim, xval)
        syms x [1 dim]
```

```
        func = (abs(2-x(1)))^3+(abs(2-x(2)))^3+(abs(2-x(3)))
           ^3+(norm(x))^2;
        g = matlabFunction(func,'vars',{x});
        value = g(xval);
        coordinate = [];
        derivative = gradient(func);
        grad = [];
        for i = 1:dim;
            coordinate = [coordinate xval(i)];
            deriv_val = matlabFunction(derivative(i), 'vars',{
               x});
            grad = [grad deriv_val(xval)];
        end
        coordinate = [coordinate value];
        grad = [grad -1];
end

%This function gives the average of the bounds to find a
%non-vertical/non-horizontal hyperplane
function avg_hyp = another_hyperplane(lower_bound, upper_bound,
    dim)
        hrep = [];
        avg = [];
        for i = 1:dim;
            val = (lower_bound(i) + upper_bound(i))/2;
            avg = [avg val];
        end
        [avg_bound] = avg;
        [coordinate gradient] = pconvex(dim, avg_bound);
        avg_hyp = [hrep; gradient coordinate*gradient'];
end

%This function gives the initial representation (cylindrical
%representation) using box constraints
```

```
function [init_hrep] = initial_representation(lower_bound,
   upper_bound, dim, hyperplane_set)
        rep.B = [];
        rep.b = [];
        for i = 1:size(hyperplane_set,1)
            rep.B = [rep.B ; hyperplane_set(i,[1: dim+1])];
            rep.b = [rep.b ; hyperplane_set(i,dim+2)];
        end
        lower_bound = [lower_bound; -inf];
        upper_bound = [upper_bound; inf];
        rep.l = lower_bound;
        rep.u = upper_bound;
        init_hrep=polyh(rep,'h');
end

%This function checks whether a vertex satisfies the
   terminiating condition
function TorF = isavertex(v1,v2,epsilon)
        TorF=0;
        for i=1:size(v2,2)
            if norm(v1-v2(:,i)) < 10^-10
               TorF=1;
               break;
            end
        end
end

%This function seperates the vertices which violate the
   terminating
%condition from those who satisfy
function [unknown_vertices] = vertex_enumeration(representation
   , known_vertices, dim)
        unknown_vertices=[];
        res=vrep(representation);
```

```
        vert=(res.V);
        for i=1:size(vert,2)
            if ~(isavertex(vert(:,i),known_vertices,dim))
                unknown_vertices=[unknown_vertices vert(:,i)];
            end
        end
end

function [xstar] = LW(vert_epi,dim)
        syms x [1 dim]
        func = (norm(x))^2;%some other convex function h;
        h = matlabFunction(func,'vars',{x});
        minimal_set = [];
        for i = 1:size(vert_epi,2)
            vert_point = vert_epi(:,i);
            x_i = [];
            for j = 1:dim
                x_i = [x_i vert_point(j)];
                r = vert_point(dim+2);
            end
            val = r-h(x_i);
            minimal_set = [minimal_set val];
        end
        [min_val, min_index] = min(minimal_set);
        opt_soln = vert_epi(:,min_index);
        xstar = opt_soln(1:dim);
end

function [init_rep xbar nvertices xstar k time bensolve_number
   f1_eval] = initial_approx(lower_bound, upper_bound, dim,
   epsilon)
        tic;
        xbar = [];
        hyperplanes = [];
```

```
init_hyp = another_hyperplane(lower_bound, upper_bound
   , dim);
hyperplanes = [hyperplanes init_hyp];
init_rep = initial_representation(lower_bound,
   upper_bound, dim, hyperplanes);
 k = 0;
 counter = 0;
 vert_call = 0;
 f1_eval = 1;
 v_known = [];
 while counter >=0
     k = k+1;
     check = 0;
     v_unknown = vertex_enumeration(init_rep, v_known,
        dim);
     vert_call = vert_call + 1;
     %v_unknown
     for i=1:size(v_unknown,2)
         vertex = v_unknown(:,i);
         new_vertex = [];
         for i = 1:dim
             new_vertex = [new_vertex vertex(i)];
         end
         %new_vertex = vertex(1:dim);
         [x dx] = pconvex(dim,new_vertex);
         f1_eval = f1_eval + 1;

         v_known = [v_known vertex];

         if abs(x(dim+1)-vertex(dim+1)) > epsilon
            new_hyp = [];
            new_hyp = [new_hyp; dx x*dx'];
```

```
                    hyperplanes = vertcat(hyperplanes, new_hyp
                        );
                    check = 1;

                else x = [x vertex(dim+1)];
                    xbar = horzcat(xbar, x');
                end
            end
            if check == 0
                break
            end
            init_rep = initial_representation(lower_bound,
                upper_bound, dim, hyperplanes);
        end
        nvertices = size(xbar,2);
        xstar = LW(xbar,dim);
        k = k-1;
        bensolve_number = vert_call;
        time = toc;
end
```

B.3 Code for Algorithm 3

```
lb = ; %lower bounds
ub = ; %upper bounds
dimension = n;
epsilon = ; %epsilon value

[init_rep iterations x_eps time vert_func_call first_func_call]
    = initial_approx(lb,ub,dimension,epsilon) %Outputs = time,
   vertices

if dimension < 3
```

```
    plot(init_rep);
else
    disp("Dimension too high; cannot plot the polyhedral
       function")
end

%This function evaluates function values at the x points
function [coordinate grad] = pconvex(dim, xval)
        syms x [1 dim]
        func = 1.03*(x(1)^2+x(2)^2)-cos(x(1))*cos(x(2));
        g = matlabFunction(func,'vars',{x});
        value = g(xval);
        coordinate = [];
        derivative = gradient(func);
        grad = [];
        for i = 1:dim;
            coordinate = [coordinate xval(i)];
            deriv_val = matlabFunction(derivative(i), 'vars',{
               x});
            grad = [grad deriv_val(xval)];
        end
        coordinate = [coordinate value];
        grad = [grad -1];
end

%This function gives the average of the bounds to find a
%non-vertical/non-horizontal hyperplane
function avg_hyp = another_hyperplane(lower_bound, upper_bound,
    dim)
        hrep = [];
        avg = [];
        for i = 1:dim;
            val = (lower_bound(i) + upper_bound(i))/2;
            avg = [avg val];
```

```
        end
        [avg_bound] = avg;
        [coordinate gradient] = pconvex(dim, avg_bound);
        avg_hyp = [hrep; gradient coordinate*gradient'];
end

%This function gives the initial representation (cylindrical
%representation) using box constraints
function [init_hrep] = initial_representation(lower_bound,
   upper_bound, dim, hyperplane_set)
        rep.B = [];
        rep.b = [];
        for i = 1:size(hyperplane_set,1)
            rep.B = [rep.B ; hyperplane_set(i,[1: dim+1])];
            rep.b = [rep.b ; hyperplane_set(i,dim+2)];
        end
        lower_bound = [lower_bound; -inf];
        upper_bound = [upper_bound; inf];
        rep.l = lower_bound;
        rep.u = upper_bound;
        init_hrep=polyh(rep,'h');
end

%This function checks whether a vertex satisfies the
   terminiating condition
function TorF = isavertex(v1,v2,epsilon)
        TorF=0;
        for i=1:size(v2,2)
            if norm(v1-v2(:,i)) < 10^-10
               TorF=1;
               break;
            end
        end
end
```

```
%This function seperates the vertices which violate the
   terminating
%condition from those who satisfy
function [unknown_vertices] = vertex_enumeration(representation
   ,known_vertices,epsilon)
        unknown_vertices=[];
        res=vrep(representation);
        vert=(res.V);
        for i=1:size(vert,2);
            if ~isavertex(vert(:,i),known_vertices,epsilon)
                unknown_vertices=[unknown_vertices vert(:,i)];
            end
        end
end

function [xstar] = LW(vert_epi,dim)
        syms x [1 dim]
        func = x(1)^2+x(2)^2;%some other convex function h;
        h = matlabFunction(func,'vars',{x});
        minimal_set = [];
        for i = 1:size(vert_epi,2)
            vert_point = vert_epi(:,i);
            x_i = [];
            for j = 1:dim
                x_i = [x_i vert_point(j)];

                r = vert_point(dim+1);
            end
            val = r-h(x_i);
            minimal_set = [minimal_set val];
        end
        [min_val, min_index] = min(minimal_set);
        xstar = vert_epi(:,min_index);
```

```
        %xstar = opt_soln(1:dim);
end

function [init_rep k xstar time bensolve_number f1_eval] =
   initial_approx(lower_bound, upper_bound, dim, epsilon)
        tic;
        hyperplanes = [];
        init_hyp = another_hyperplane(lower_bound, upper_bound
           , dim);
        hyperplanes = [hyperplanes init_hyp];
        init_rep = initial_representation(lower_bound,
           upper_bound, dim, hyperplanes);
        v_known = [];
        k = 0;
        counter = 0;
        f1_eval = 1;
        vert_call = 0;
        while counter >=0
            LWvertices = [];
            k = k + 1;
            vert = vertex_enumeration(init_rep,v_known,epsilon
               );
            vert_call = vert_call +1;
            for i = 1:size(vert,2)
               v = vert(:,i);
               LWvertices = horzcat(LWvertices,v);
            end

            [x_i] = LW(LWvertices,dim);
            xdim = [];
            for i = 1:dim
                xdim = [xdim x_i(i)];
            end
            [x dx] = pconvex(dim,xdim);
```

```
            f1_eval = f1_eval + 1;
            v_known = [v_known x_i];
            error = x(dim+1)-x_i(dim+1);
            if error <= epsilon
                final_x = x;
                k = k - 1;
                break
            else
                new_hyp = [];
                new_hyp = [new_hyp; dx x*dx'];
                hyperplanes = vertcat(hyperplanes, new_hyp);
                init_rep = initial_representation(lower_bound,
                   upper_bound,dim,hyperplanes);
            end
        end
        xstar = [final_x x_i(dim+1)];
        bensolve_number = vert_call;
        time = toc;
end
```