On the Use of Convex Underestimators in Global Optimization

Anders Skjäl

PhD Thesis in Applied Mathematics
Department of Natural Sciences
Åbo Akademi University

Åbo, Finland 2014

ISBN 978-952-12-3018-9

Painosalama Oy
Åbo 2014

Preface

I want to express my gratitude to my supervisor Prof. Tapio Westerlund for the opportunity to work in the OSE group. He has provided guidance and feedback but also the freedom to pursue research questions according to my own interests. I also want to thank Prof. Göran Högnäs and others who have educated me in mathematics.

The work on this thesis was carried out during the years 2010-2013 at the Center of Excellence in Optimization and Systems Engineering (OSE). Additional funding from the Walter Ahlström Foundation, the Academy of Finland project 127992, and the Research Institute of the Foundation for Åbo Akademi is gratefully acknowledged.

I thank Prof. Christodoulos A. Floudas at Princeton University for the opportunity to visit his research group. I also thank his former student Ruth Misener, who aided me with practicalities and spent a lot of time improving our joint articles.

I thank my current and former colleagues at Process Design and Systems Engineering, and others sharing our kitchen, for creating a good atmosphere at work. They have provided ideas and solutions to many of my queries, work-related or not. Special thanks go to some fellow members of the OSE group; Andreas Lundell, Axel Nyberg, Otto Nissfolk, Mikael Nyberg, Ray Pörn, Mikael Kurula and Toni Lastusilta.

Friends and family have provided a welcome counterweight to work. I thank Vilgot Strömsholm and Carl-Mikael Wiklund for various activities and adventures during a decade in Åbo.

Åbo, March 2014

Anders Skjäl

Svensk sammanfattning

Ett mål inom global optimering är att identifiera de bästa möjliga lösningarna med korrekt angiven noggrannhet. Det här skiljer sig från lokala optimeringsmetoder och metaheuristiker som inte kan ge något mått på kvaliteten hos funna lösningar. Algoritmer som konvergerar mot en global lösning med sannolikheten 1 kallas ibland globala, även om de inte kan avgöra vilken precision som uppnåtts. I den striktare tolkning som används i denna avhandling bör en global metod ge en lösning med förangiven noggrannhet i ändlig tid. Detta forskningsområde benämns ibland *deterministisk global optimering*.

Fokus i avhandlingen ligger på icke-linjära problem med ett ändligt antal kontinuerliga och diskreta variabler. De vedertagna akronymerna för den här typen av optimering är NLP (nonlinear programming) och MINLP (mixed-integer nonlinear programming). För att förbigå vissa olösliga problem kan det antas att variablerna är begränsade till ändliga intervall. Ingen känd algoritm kan effektivt lösa varje problem av den här typen. Om en välkänd förmodan inom datavetenskap stämmer, att två centrala komplexitetsklasser är olika, så medför det att ingen sådan algoritm är möjlig på en klassisk dator. Kvantdatorer antas inte heller kunna lösa alla dessa problem effektivt.

Om ett optimeringsproblem har någon speciell struktur så kan en skräddarsydd algoritm fungera bäst. Ibland lyckas man då avgränsa en klass av problem sådan att alla exempel kan lösas effektivt. En annan möjlighet är att söka efter algoritmer som fungerar ”i praktiken” utan effektivitetsgarantier i de värsta fallen. Simplexmetoden för linjär programmering faller inom den här kategorin. Den utför i värsta fall ett exponentiellt antal steg, men de fallen är så atypiska att de inte räcker som orsak för att välja garanterat polynomiella inrepunktsmetoder.

I vissa fall uppnår man önskad noggrannhet med välavvägd modellering och en effektiv lösare. För många problem av praktiskt intresse tvingas globala optimeringsalgoritmer ändå förgrena sökningen till stora sökträd. En fördel med så kallade branch-and-bound-metoder (ungefär dela-och-begränsa) är att de håller reda på konvergerande övre och nedre gränser för den globala lösningen. Om körningen avbryts efter en viss tid kan den bästa kända lösningen med dittills verifierade felgränser tas tillvara. Tillämpningen avgör om global optimering av det här slaget är ett lämpligt tillvägagångssätt.

För att använda branch-and-bound-metoder krävs att man kan beräkna giltiga undre gränser vid minimering (övre gränser vid maximering). Det här görs genom förenk-

lingar som relaxering av heltalsvariabler och besvärliga villkor. Konvex underskattning av funktioner är en viktig sådan förenkling. I vissa fall kan funktionens konvexa envelopp, den största konvexa underskattaren, användas. Användbara representationer av konvexa envelopper är kända för många enkla funktioner.

I allmänhet är det ändå ofta svårt att beskriva enveloppen och istället används metoder för att konstruera *någon* konvex underskattare. Två välkända sådana metoder beskrivs i kapitel 4. Den ena bygger på att dela upp en funktion i enkla komponenter såsom addition, multiplikation och elementära funktioner. Komponenterna kan sedan successivt relaxeras med specifika metoder, resultatet är en konvex underskattare av den ursprungliga funktionen. Den andra metoden är känd som αBB ('BB' som i branch-and-bound). Metoden beskrevs som en fullständig optimeringsalgoritm, men beteckningen används också för den specifika underskattningsmetod som ingår. Genom att applicera andra ordningens konvexitetsvillkor på en intervallapproximation av Hessematrisen kan man bestämma en enkel additiv perturbation som konvexifierar funktionen.

Mina forskningsbidrag har inriktat sig på att studera och förbättra αBB-metoden. Tillsammans med mina medförfattare har jag beskrivit hur metoden kan utvidgas till att inkludera bilinjära perturbationer för att få tätare underskattare. På grund av formen på perturbationernas Hessematriser har vi kallat utvidgningen *icke-diagonal* αBB. Jag har även studerat nya metoder för att bestämma parametrar för diagonala och icke-diagonala αBB-underskattare. Metoderna har jämförts teoretiskt och experimentellt, både mot varandra och mot den mest mångsidiga av tidigare beskrivna metoder. I en konferensartikel jämför jag αBB med en specialiserad metod för konvex underskattning av polynom.

Kapitel 2–4 fungerar som en bred introduktion till global optimering. Jag tar upp komplexitetsteori som en nyttig bakgrund för att förstå approximationsalgoritmer och hur global optimering förhåller sig till andra optimeringsparadigm. I kapitel 3 diskuteras globala lösares pålitlighet och huvudstegen i en branch-and-bound-algoritm. Kapitel 4 fokuserar på konvex underskattning, den specifika tillämpningen för min forskning. Kapitel 5 innehåller kommentarer till och korta sammanfattningar av artiklarna i avhandlingen. I kapitel 6 diskuteras globala lösare på ett allmänt plan och aspekter som ännu borde undersökas när det gäller αBB-metoder.

Contents

Chapter 1

Introduction

Global optimization, in a strict sense, is concerned with finding a nearly global solution with verified error bounds. For general problems, without some simplifying structure, this will usually involve branch-and-bound methods. In branch-and-bound algorithms a problem is repeatedly divided into subdomains, and approximated with simpler problems that can be solved with less effort. The αBB method provides such simplifications in the form of convex underestimators of smooth functions.

The aim of this thesis is to further the understanding of αBB-type underestimators and to investigate their limits. For any given smooth function on a box domain there exists an optimal αBB underestimator, yielding the smallest possible approximation error. To calculate an optimal underestimator is usually impractical and it may be a harder problem than minimizing the function itself. A more practical approach is to resort to two big simplifications, interval arithmetic and sufficient convexity conditions. The elements of the Hessian need to be approximated with some constant numbers or intervals, because the unsimplified convexity condition (see Section 4.2)

$$\forall \boldsymbol{x} \in D: \; H_f(\boldsymbol{x}) + H_P \succeq 0$$

lies in the realm of semi-infinite programming. Even with the Hessian approximated by intervals, there are no equivalent conditions for convexity that can be efficiently checked and used to calculate αBB parameters. Instead, one must use sufficient conditions which are sometimes unnecessarily conservative.

Improvements in interval approximations and convexity conditions for interval matrices could potentially increase the usefulness of αBB underestimators. Interval arithmetic is a well-studied area with applications far beyond αBB, the focus in my research is on convexity conditions. Reading the existing αBB research, I gradually realized how various sufficient conditions could be applied to determine αBB parameters. I also had the opportunity to participate in developing the idea of "nondiagonal" αBB perturbations, which had so far only been hinted at in a short conference paper (see Section 5.2). This extension, in turn, asked for nondiagonal parameter calculation methods to be developed and tested.

The terminology used above is defined in subsequent chapters.

1.1 Structure of the Thesis

The research for this thesis appears in the five included publications. Especially Papers II and V are relatively self-contained. In this summary part of the thesis I provide perspective on convex underestimators and the global optimization paradigm, to show how they fit into a broader context of computational problem solving. The chapters proceed from broader views to the specific field of my contributions.

Chapter 2 presents the basics of algorithm analysis and complexity theory. This is a perspective that is relevant to anyone solving problems using computers. Below the layers of abstraction and terminology that may follow with a specific application field, the computational performance is determined by the number of basic operations an algorithm must perform, like arithmetic operations on data representing integers or floating-point numbers. In practice larger units of work are often useful. In mathematical programming the same functions may be evaluated repeatedly to check feasibility or the objective value. Sometimes these evaluations require, for example, calling expensive black-box functions or solving differential equations. Then the rest of the algorithm has a nearly negligible effect on the time performance and it makes sense to count in terms of function evaluations.

A specific algorithm can be analyzed to determine the worst-case and sometimes the average-case performance. It is also possible to study problems independently of the solution algorithms, sometimes one can infer performance bounds that must hold for any algorithm aimed at the problem. The algorithmic task may be to find an exact answer or, more commonly with practical optimization problems, to determine a solution within a tolerance of the global optimum.

There is, perhaps, a gap between complexity theory and the practice of nonlinear optimization. If a model has scientific or economical interest it is not enough to show that the general problem is hard, a solution must be attempted. Nevertheless, complexity theory is a convenient language to describe the intrinsic hardness of classes of optimization problems. It is also a gateway to a broad reference literature on the best known approximation algorithms.

The basic goal of global optimization is defined in Chapter 3: reliable algorithms that output a verified global optimum or prove the infeasibility of the input problem. Branch-and-bound is a key component in more general nonlinear programming algorithms with fewer assumptions on the structure of the problem. The typical steps of a branch-and-bound algorithm are explained.

The construction of convex underestimators and their application in nonlinear programming is described in Chapter 4. Some examples of convex envelopes and other tight underestimators from the literature are reviewed. It seems reasonable to contrast αBB with another prominent and highly succesful method for underestimating generic expressions; αBB and factorable program relaxations are described in Section 4.2.

Chapter 5 contains notes on the included articles, Papers I–V listed below. The comments describe the context where the research was made and the author's contributions are stated at the end of the chapter. The maturity level of αBB methods and global

solvers in general are discussed in the conclusions in Chapter 6.

1.2 List of Publications

Paper I A. Skjäl, A. Lundell, and T. Westerlund. Global optimization with C^2 constraints by convex reformulations. *Chemical Engineering Transactions*, Volume 24, pp. 373–378, 2011.

Paper II A. Skjäl, T. Westerlund, R. Misener, and C. A. Floudas. A generalization of the classical αBB convex underestimation via diagonal and non-diagonal quadratic terms. *Journal of Optimization Theory and Applications*, 154(2):462–490, 2012.

Paper III A. Skjäl, R. Misener, T. Westerlund, and C. A. Floudas. A generalization of classical αBB underestimation to include bilinear terms. In I. D. L. Bogle and M. Fairweather, editors, *22nd European Symposium on Computer Aided Process Engineering*, Volume 30 of *Computer Aided Chemical Engineering*, pp. 1202–1206. Elsevier, 2012.

Paper IV A. Skjäl and T. Westerlund. A comparison of two convex underestimation methods for quadratic functions. In A. Kraslawski and I. Turunen, editors, *23rd European Symposium on Computer Aided Process Engineering*, Volume 32 of *Computer Aided Chemical Engineering*, pp. 535–540. Elsevier, 2013.

Paper V A. Skjäl and T. Westerlund. New methods for calculating αBB-type underestimators. *Journal of Global Optimization*, 58(3):411–427, 2014.

Apart from the included publications, the author made contributions to a paper on piecewise convex reformulations:

[75] A. Lundell, A. Skjäl, and T. Westerlund. A reformulation framework for global optimization. *Journal of Global Optimization*, 57(1):115–141, 2013.

Chapter 2

Complexity Theory and Algorithms

The central goal of computational complexity theory is to find absolute bounds on the difficulty of problems. Theoretical computer models are used to facilitate formal proofs. The use of very simple models, such as Turing machines, is justified since other reasonable computer models can be simulated with only a polynomial slowdown. The implications for real-world computers are fundamental.

Complexity theory is a wide and active field of research. The aim of this chapter is to present some results relevant to the optimization problems discussed in the rest of the thesis. The choice of contents is inspired by several textbooks in the field. The influential textbook by Garey and Johnson [42] describes the milestones before 1979 and contains an extensive appendix listing known NP-complete problems. Ausiello et al. [15] present the theory of approximation algorithms and list approximability properties, "good news/bad news", for specific problems in an appendix. Vazirani [121] collects some of the best available approximation algorithms. Wegener [124] covers a wide selection of topics from complexity theory.

2.1 Algorithm Runtimes

The analysis of algorithms is closely related to complexity theory. An algorithm provides a constructive upper bound on the hardness of a problem. An algorithm can be analyzed in terms of how the number of operations and the amount of storage space needed depends on the size of the input. It is usually clear how operations are counted. In numerical algorithms one might for example count the number of floating-point operations, since they are expensive compared to other basic instructions. The increasing sizes of random-access memory and caches have lessened the importance of space complexity. Only time complexity is discussed in this chapter.

The efficiency of an algorithm is often given as the worst-case asymptotic runtime of a class of problems. Asymptotic runtimes provide a way to concentrate on the essential complexity of algorithms, ignoring overheads and faster subalgorithms for pre- or post-processing. The asymptotic behavior will often dominate even for moderately sized problem instances. The *big O* notation is commonly used to state the asymp-

totic performance. Consider an algorithm for a specific problem. Let $T(n)$ denote the largest number of steps taken for any problem instance with input size n. We write $T(n) = O(f(n))$ if there exists positive constants M and n_0 such that

$$T(n) \leq M|f(n)| \text{ for all } n > n_0.$$

Note that $T(n) = O(f(n))$ is actually a set relation, some authors prefer the notation $T(n) \in O(f(n))$. Other notations are sometimes used to specify asymptotic lower bounds (Ω) and asymptotic equivalence (Θ).

Two well-known tasks, the calculation of discrete Fourier transforms and matrix products, serve to illustrate some points about algorithm efficiency.

The discrete Fourier transfom of a sequence of length n is commonly calculated with the algorithm known as the fast Fourier transform (FFT). The algorithm uses only $O(n \log n)$ operations whereas the naïve method has $O(n^2)$ runtime. Cooley and Tukey published the method in 1965 [30] but restricted versions had been described before. It turns out that Gauss found and used the general algorithm around 1805, see Heideman et al. [54] for a historic account. The efficiency improvement was important, Heideman et al. describe it as *"a turning point in digital signal processing and certain areas of numerical analysis."*

Matrix multiplication of $n \times n$ matrices is an important operation by itself and as a subalgorithm for other tasks, like matrix inversion. The naïve approach, calculating all elements separately as dot products, is an $O(n^3)$ algorithm. On the other hand, all elements in the output must be calculated, so $\Omega(n^2)$ is a definitive lower bound on any algorithm. Strassen published a method in 1969 [110] with an asymptotic performance of $O(n^{2.81})$. Strassen's algorithm is thus asymptotically faster than the basic algorithm. In practice, the memory architecture of the computer will partly determine where the asymptotic behavior dominates. Linear algebra libraries can be fine-tuned to get good performance on a specific processor. D'Alberto and Nicolau [35] show that the crossover size where Strassen's algorithm becomes more effective is quite large, above $n = 1000$ for many systems.

Coppersmith and Winograd [31] presented an algorithm which further improves the time complexity to $O(n^{2.38})$. Since then there has only been slight improvements in asymptotic runtime. The Coppersmith-Winograd algorithm is mostly of theoretical interest, since the crossover size would be enormous [97]. Matrices with a special structure or sparseness may allow faster specialized multiplication algorithms. Such matrices form easier subclasses of the general problem of matrix multiplication.

The algorithms mentioned in this section have polynomial upper bounds on their asymptotic runtimes ($n \log n < n^2$). This distinguishes the underlying problems from problems which do not have such algorithms. Polynomial algorithm improvements such as the FFT described above are important. Nevertheless, significant insights have been gained in computer science by adopting a broader perspective where all polynomial algorithms are considered efficient. The problems are then classified into *tractable* problems with some polynomial algorithm and *intractable* problems without such algorithms.

2.2 Decision Complexity

There are different types of algorithmic problems. Decision problems are problems with a 'yes' or 'no' answer. Optimization problems ask for one optimal solution to a problem. Evaluation problems ask for the objective value of an optimal solution. Most optimization problems have natural decision and evaluation variants, "Is there a solution with an objective value better than K?" and "Find the value of an optimal solution", respectively.

The solution to the optimization problem will quickly answer the decision and evaluation problems, assuming that the objective can be evaluated easily in a point. Wegener [124] shows a partial converse to this, namely that the three variants are (Turing)-equivalent under conditions satisfied by many discrete problems. Some basic concepts are defined in terms of decision problems. There are implications for general optimization problems since optimization is at least as hard as the related decision problem.

The class of all decision problems solvable in worst-case polynomial time is denoted P. A formal definition can be given in terms of languages and Turing machines, see Garey and Johnson [42]. Some problems are provably harder than polynomial time. Consider, for example, a guessing game where an n-digit number (base 2) should be found, without any feedback like "smaller/greater". To find the correct number will take $O(2^n)$ guesses in both the worst-case and on average. This needle-in-the-haystack problem, see Wegener [124], can only be solved for relatively small n. Another inherently hard problem is to check if white has a winning strategy in chess, generalized to a game on an $n \times n$ board [41]. Many problems of interest are somewhere in between; no polynomial time algorithm is known but it has not been proven that no such algorithm exists.

A decision problem belongs to the complexity class NP if there exists a *witness* (a certificate, a proof) that can be checked in polynomial time for every 'yes' instance of the problem. The class abbreviation stands for *nondeterministic polynomial time*, stemming from a characterization through nondeterministic Turing machines. As an example, the bin packing decision problem belongs to NP. If the objects can be packed in K bins, then a proof of this is simply a partition satisfying the bound. It can be checked in linear time that the partition satisfies the conditions. Similar reasoning shows that traveling salesperson (TSP), knapsack, satisfiability, Hamiltonian cycle and many other problems belong to NP. Proving that a problem belongs to NP is usually easy.

'Yes' instances of an NP problem can be solved by guessing a proof, 'no' instances by trying all possible proofs. This is not a practical approach, it is rather an intuitive characterization of a class of problems. Note that there is an asymmetry between 'yes' and 'no' instances in the definition of NP. The class of problems with witnesses for every 'no' instance is named coNP.

The class P is a subclass of NP. To check a 'yes' answer to a problem in P the verifier can solve the problem itself in polynomial time. A central question in computer science is whether NP is larger than P or not. It has become known as the P *versus* NP *problem* and is one of the seven Millennium Prize Problems. Most experts in the field lean

towards the conjecture P ≠ NP [55]. Many theorems about complexity classes require the assumption P ≠ NP or are trivial if it fails.

If P ≠ NP, then the class NP – P has interesting structure. One tool for studying this structure is reductions. A *polynomial time reduction* from problem A to problem B is a polynomial time function mapping instances of A to instances of B with the same 'yes' or 'no' answer. Such reductions give a partial order between problems. The notation $A \leq_p B$ is used if decision problem A can be reduced to B. The existence of such a reduction means that, from a polynomial time perspective, B is at least as hard as A. If $A \leq_p B$ and $B \leq_p A$, then the problems are *polynomially equivalent*, $A \equiv_p B$. This is an equivalence relation on decision problems.

With these definitions it makes sense to ask if there are problems as hard as any problem in NP. A decision problem A is called NP-*hard* if $B \leq_p A$ for all $B \in$ NP. If the problem is NP-hard and, in addition, belongs to NP it is called NP-*complete*. The corresponding class notation is NPC. It is not obvious that NP-complete problems exist, but Cook showed in 1971 that the satisfiability problem is NP-complete [29]. Levin made similar discoveries independently [115] and Karp listed 21 NP-complete problems [63] in 1972. After these pioneering works it is much easier to prove NP-hardness of a problem. It suffices to describe a reduction from one of the known NP-complete problems to the new problem. Today a multitude of problems are known to be NP-complete.

Garey and Johnson [42] list three common methods of constructing a reduction: restriction, local replacement and component design. A proof by restriction is simply noting that a special case of the problem is NP-complete by itself. For example, mixed-integer programming contains 0-1 integer programming as a special case, one of Karp's 21 problems, and therefore mixed-integer programming is NP-hard. Local replacement works by transforming basic units of an NP-complete problem to one or a few basic units in the new problem. The satisfiability problem can be reduced to 3-satisfiability, where all clauses have length 3. After writing the boolean function in conjunctive normal form, any clauses longer than 3 can be replaced in a systematic way by shorter clauses with auxiliary variables:

$$x_1 + x_2 + x_3 + x_4 + x_5 \;\rightarrow\; (x_1 + x_2 + y_1)(\overline{y}_1 + x_3 + y_2)(\overline{y}_2 + x_4 + x_5).$$

Component design is more involved than local replacement; the transformations are more complex and their influence may be nonlocal. An entertaining construction by Kaye reduces the satisfiability problem to circuits on a Minesweeper board, thereby proving NP-completeness of certain aspects of Minesweeper [64]. Scott et al. commented and extended the results [105].

Most studied problems in NP have been shown to be solvable in polynomial time or NP-complete. Ladner showed that if P ≠ NP, then the intermediate class NPI = NP – (P ∪ NPC) is nonempty. Integer factorization and graph isomorphism checking are possible NPI-candidates [124].

Many problems with important practical applications are NP-complete. The NP-completeness of a problem indicates that no polynomial time algorithm can be found

in the literature. Such an algorithm would prove, somewhat unexpectedly, that P = NP, and the accomplishment would be well-known.

2.3 Optimization Complexity

Asking for an exact optimal solution to an optimization problem is often unnecessarily complicated. A "nearly global" solution is acceptable in many applications. The possibility of calculating good approximate solutions in polynomial time is also studied in complexity theory.

A useful measure of the quality of a solution is the relative difference in value compared to an optimal solution. Let x^* denote an optimal solution and x a feasible solution to an optimization problem. Depending on the problem x may be a point in $\mathbb{Z}^d$ or $\mathbb{R}^d$, or some other structure, like a Hamiltonian cycle in the case of a TSP. Assume that the objective function $f > 0$. The quality of the solution x is described by approximation ratios, $r(x) = f(x)/f(x^*)$ for minimization problems and $r(x) = f(x^*)/f(x)$ for maximization problems. Let ε be a relative error tolerance. A solution is called *ε-optimal* if $r(x) \leq 1+\varepsilon$.

Relative sizes and errors are usually more meaningful than their absolute counterparts. For example, a unit conversion does not change the relative measures. The positivity assumption on the objective function is valid in many models. Analogous definitions can be stated for strictly negative objective functions. For models where the objective value 0 has no specific significance one may consider adding a constant to the objective or relating the error to the interval width $\max f(x) - \min f(x)$ [119].

Many optimization problems, despite having NP-hard decision variants, allow the calculation of approximate solutions in polynomial time. Problems with a polynomial time algorithm for calculating an ε-optimal solution, for some fixed $\varepsilon \geq 0$, belong to the complexity class APX ("approximable"). If an algorithm can take ε as an additional input and give an ε-optimal solution in polynomial time with respect to the input size, it is called a *polynomial time approximation scheme*. Problems allowing such schemes belong to the class PTAS.

A polynomial time approximation scheme does not guarantee the practicability of finding high accuracy solutions, because the asymptotic runtime may grow quickly with $1/\varepsilon$, *e.g.*, as $O(n^2 \cdot 2^{1/\varepsilon})$. A scheme where the asymptotic runtime is polynomial also in $1/\varepsilon$ is called a *fully polynomial time approximation scheme*. The corresponding complexity class is FPTAS. It follows from the class definitions that FPTAS $\subseteq$ PTAS $\subseteq$ APX. The subset relations are proper if P $\neq$ NP.

The traveling salesperson problem and some restricted versions of the problem have different approximability properties. A broad set of references can be found in the appendix of [15]. If P $\neq$ NP, then the general TSP is not approximable within a constant factor ($\notin$ APX). This can be shown by noting that such an algorithm could be modified to check the existence of Hamiltonian cycles, an NP-complete decision problem, in polynomial time. When the distances in the problem correspond to a metric, Christofides's $O(n^3)$ algorithm [28] finds an approximate solution with $\varepsilon = 1/2$. Papadimitriou and Vempala proved that no approximation algorithm can achieve $\varepsilon < 1/219$ unless P = NP

[90]. Arora described a PTAS for TSP with distances corresponding to an embedding in $\mathbb{R}^d$ with an l^p ($p \geq 1$) norm [13]. The asymptotic runtime of the algorithm in two dimensions is $O\left(n^3(\log n)^{O(1/\varepsilon)}\right)$. The expression grows faster than any polynomial in $1/\varepsilon$ so the algorithm is not fully polynomial. The optimization version of the knapsack problem allows a fully polynomial time approximation scheme with runtime $O(n^3/\varepsilon)$ [124].

A described and analyzed algorithm sets an upper bound on the approximability properties of a problem. Lower bounds on the efficiency of any approximation algorithm for the same problem can be established under the P ≠ NP assumption.

The *gap technique* relies on creating a related NP-hard decision problem which proves the hardness of the approximation problem. Assume that the range of the optimization objective is a subset of the disjoint union $(0, a] \cup [b, +\infty)$. If it is NP-hard to decide which interval the optimal value lies in, then it is also hard to find approximate solutions with $\varepsilon = (b - a)/a$. That is, no polynomial time algorithm exists unless P = NP. The notion NP-hard is extended to all algorithmic problems by replacing the $\leq_p$ reduction with the more general polynomial time *Turing reduction* $\leq_T$ [124].

It is fruitful to define a more detailed reduction between optimization problems. A PTAS-reduction from optimization problem A to problem B, $\leq_{\text{PTAS}}$, is a triple of polynomial time functions (f, g, α). Instances of A are mapped by f to instances of B, and an approximate solution is mapped back by g. The mappings are such that if the solution to the B instance is $\alpha(\varepsilon)$-optimal, then the A instance is ε-optimal. Assume that $A \leq_{\text{PTAS}} B$. Then $B \in$ PTAS implies $A \in$ PTAS and, similarly, $B \in$ APX implies $A \in$ APX. If $\alpha(\varepsilon)$-optimal solutions to problem B can be found in polynomial time, then so can ε-optimal solutions to problem A. The contrapositive statements are also useful.

The PCP Theorem from 1992 by Arora et al. [14] facilitated the use of the gap technique. The hardness of approximation has since been established for many well-known problems. PCP is an acronym for Probabilistically Checkable Proof, a concept where a verifier with bounded resources can be convinced to accept all 'yes' instances of a decision problem and to reject 'no' instances' with a high probability. PCP$(r(n), q(n))$ is the class of problems with proofs that can be probabilistically checked in polynomial time, using $O(r(n))$ random bits and reading $O(q(n))$ bits of the proof. The PCP Theorem gives the surprising characterization NP = PCP$(\log n, 1)$. For a description of applications and related ideas, see the textbook by Goldreich [46].

2.4 Intractability in Global Optimization

Most problems studied in textbooks on complexity theory have a distinctly combinatorial character. Global optimization is also concerned with continuous and mixed-integer models. Mixed-integer programming is NP-hard, since many NP-complete combinatorial problems are easily modeled as binary programming. The references in this section show that continuous and mixed-integer optimization are amenable to complexity theory-based investigation.

Convex problems are solvable in polynomial time under mild conditions, see [21, 86] or the lecture notes by Nemirovski [85]. There is reason to consider most continuous convex problems as easy. Rockafellar [98] expressed this as: *"the great watershed in optimization isn't between linearity and nonlinearity, but convexity and nonconvexity."*

Difficulties may appear even with the simplest imaginable nonconvexities. Quadratic programming as a decision problem has been proven NP-complete [104, 118]. Even properties like local optimality are NP-hard to check [84, 91]. Pardalos and Vavasis [92] showed that one negative eigenvalue of the objective function is enough to make the problem NP-hard (but this case admits an FPTAS [119]). Bellare and Rogaway [18] proved that if P ≠ NP, then quadratic programming is not in PTAS, and under a stronger assumption the problem is not even in APX. Rohn [100] has gathered algorithms and complexity bounds for questions concerning interval matrices. Köppe [67] surveys complexity and approximability results in nonlinear mixed-integer optimization. Ahmadi et al. recently proved that it is NP-hard to determine if a quartic polynomial is convex [5].

Some examples to the opposite also exist, nonconvex models which look similar to the NP-hard problems but turn out to be efficiently solvable. Vavasis [120] describes fractional linear programming and quadratic programming with a sphere constraint $\boldsymbol{x}^T\boldsymbol{x} \leq 1$. Geometric programming problems can be written in convex form by variable changes of the form $y_i = \log x_i$, see the tutorial by Boyd et al. [25]. Baes et al. [16] describe a black-box algorithm for minimizing a strongly convex function over the integer points in a polytope; the method runs in polynomial time for some special representations of polytopes.

2.5 Coping with Intractability

Hard problems abound in the application fields for optimization (NP-hard and otherwise). Therefore it is good to have a broad array of tools to tackle instances or classes of such problems. Much depends on the type of application: Should many similar problems be solved (scheduling, routing) or only a single instance (some allocation problems)? Is there time and hardware limitations on the solution process or will significant resources be devoted? A reasonable step is to classify the problem and see what identifiable structures it possesses. This may tell what difficulties to expect and provide a starting point for a literature search.

Local methods and heuristics can often be quickly employed to find some feasible solution. Hromkovič [61] defines a heuristic as: *"a robust technique for the design of randomized algorithms for optimization problems [...] not able to guarantee at once the efficiency and the quality of the computed feasible solution"*. To find some solution may be useful even if the quality is not guaranteed. At the least it sets a bound on the global optimum.

A strength and weakness of many heuristics is their generality. Metaheuristic techniques like simulated annealing and evolutionary algorithms need only a black-box representation of the problem. The drawback is that they may not utilize special structure

of the problem efficiently. Wegener [123] discusses runtimes and success probabilities of evolutionary algorithms and points out that algorithms designed for specific problems should in general outperform black-box search heuristics.

In contrast, approximation methods come with guarantees on the information and the asymptotic runtimes. Some problems have efficient polynomial time approximation algorithms for the required error tolerance. In other cases approximation algorithms give quickly computable bounds on the global optimum, for example first-fit algorithms for bin packing [42] or Goemans and Williamson's semidefinite programming algorithm for the maximum cut problem [45].

Deterministic global optimization algorithms should ideally produce guaranteed ε-optimal solutions for any specified tolerance. Polynomial worst-case complexity is not demanded from these algorithms, since they tackle generally hard problems. If all instances appearing in practice can be solved efficiently, then all is well. Failing this, the algorithm can be run as long as possible and keep track of the best solution and bounds established.

Hromkovič [61] discusses the combination of different techniques and the practical realities of an applied optimization project. He emphasizes how the project constraints on time and money are important factors when deciding what approaches to use. There must also be a realistic analysis of how the quality of solutions may affect profits (or other project outcomes).

A valuable skill for an analyst is to be able to make well-judged model simplifications. For models based on first principles in the natural sciences (*e.g.* chemistry, physics, molecular biology) the global optimum may have special significance as the actually occurring state [40]. This makes it harder to simplify models without destroying their value. On the other hand, if the model is purely phenomenological it is reasonable to consider more efficiently solvable models with similar descriptive power.

Chapter 3

Global Optimization

Consider the abstract form of an optimization problem:

$$\text{Find} \quad \boldsymbol{x}^* = \underset{\boldsymbol{x} \in X}{\operatorname{argmin}} f(\boldsymbol{x}). \tag{3.1}$$

Depending on the choice of function f and set X, the abstract problem in (3.1) can describe most types of optimization problems. The 'arg' syntax is often dropped in model descriptions and it is implicitly assumed that an optimal argument, corresponding to an optimal value, is determined.

The field of global optimization is, vaguely speaking, concerned with finding the absolutely best solution. Some standard definitions of different types of optimality are the following: A point $\boldsymbol{x}$ is called a *feasible solution* if $\boldsymbol{x} \in X$ and $f(\boldsymbol{x})$ is finite-valued. A feasible solution $\boldsymbol{x}$ is *locally optimal* if $f(\boldsymbol{x}) \leq f(\boldsymbol{y})$ for all $\boldsymbol{y} \in X \cap D$, for some neighborhood D of $\boldsymbol{x}$. A feasible solution $\boldsymbol{x}$ is *globally optimal* if $f(\boldsymbol{x}) \leq f(\boldsymbol{y})$ for all $\boldsymbol{y} \in X$. Let $\varepsilon > 0$ be a specified tolerance. A feasible solution $\boldsymbol{x}$ is *globally ε-optimal* if $f(\boldsymbol{x}) - \varepsilon \leq f(\boldsymbol{y})$ for all $\boldsymbol{y} \in X$. Note that ε denotes an absolute error tolerance in this chapter whereas relative errors are more common in complexity theory (Chapter 2). Both relative and absolute tolerances may be present as stopping criteria in practice.

3.1 Provably Global Solutions

Local optimization algorithms guarantee global optimality only under some assumptions, often linearity or some form of convexity. When the assumptions are invalid the same methods may output locally optimal solutions, feasible solutions or no solution at all. Greenberg's handbook [48] contains numerous counterexamples of methods failing when algorithm assumptions are not satisfied.

This does not diminish the value of local and heuristic algorithms. One solution is better than no solution and local algorithms are often fast and relatively robust. Local methods combined with techniques like multi-start [76] may tend asymptotically towards the global optimum in a stochastic sense. Linear or convex subproblems are also a part of many global algorithms, therefore the availability of fast and reliable local solvers is important.

Global optimization algorithms are designed to find globally optimal solutions, or prove infeasibility. Numerical considerations may force the user to accept a globally ε-optimal solution, especially if some decision variables are continuous. Some related algorithmic tasks are also studied within the field of global optimization. Floudas lists determination of bounds for the global optimum and enclosing all feasible solutions [40].

Note that the term "global optimization" is sometimes applied to stochastic methods which will find the global optimum with some probability. In this thesis the discussion is restricted to so called "deterministic global optimization", where the aim is essentially deterministic algorithms which find a global ε-optimum in a finite number of steps.

Neumaier suggests a classification of algorithms into incomplete, asymptotically complete, complete, and rigorous methods [88]. The last two categories are global in the sense used here. *Complete* methods are guaranteed to find a global ε-optimum under the assumption that all computations are exact. The assumption is not valid when using floating-point arithmetic, and rounding errors sometimes cause these methods to fail. *Rigorous* methods are resistant to rounding errors and will always find a global ε-optimum, with the exception of near-degenerate cases where tolerances may be exceeded.

A computer-generated result should be trusted as far as one trusts the analysis of the algorithm, the computer implementation, the correct execution of the program, the correctness of the input and its adherence to the assumptions of the algorithm. Debates on whether computer-assisted proofs are trustworthy have been raised by, *e.g.*, Appel and Haken's famous proof of the four color theorem and more recently by Hales's proof of the Kepler conjecture about sphere packing [11, 47, 51, 52].

The goal of reliable computing is common for pure and applied mathematics. It is extremely hard to produce large computer programs without bugs. These can be anything from benign memory leaks to interacting errors that corrupt the calculations in unexpected and complicated ways. For this reason computer-assisted proofs of mathematical theorems are thoroughly scrutinized. Hales's proof was accepted after a four-year peer review with the reservation that the reviewers could only be 99% certain of the correctness [111]. Some mathematicians and computer scientists consider formalization and automated proof checking as the way to trustworthy proofs. For an introduction to these efforts, see the QED Manifesto [26] and Wiedijk's overview [129].

Aside from the question of bugs, few global solvers are designed to be rigorous in the sense of Neumaier's definition [88] and will sometimes fail because of rounding errors. The demand for total reliability is perhaps slightly lower for applied analysts. They do not always need the guarantee of global optimality, and the plausibility of the optimality claim can sometimes be checked via knowledge of the specific application. The optimization of an applied model is at any rate part of the bigger work cycle of modeling, optimization, verification and validation, synthesis, and feedback between these steps.

3.2 Problem Structure

The abstract problem (3.1) is too general for the methods that are the main focus of this thesis. The discussion is now restricted to finite-dimensional nonlinear programming (NLP) with the feasible set X defined by inequality constraints:

$$\begin{array}{lll} \text{minimize} & f_0(\boldsymbol{x}) & \\ \text{subject to} & f_k(\boldsymbol{x}) \leq 0, & k \in \{1,2,\ldots,m\} \\ & x_i^L \leq x_i \leq x_i^U, & i \in \{1,2,\ldots,n\}. \end{array} \tag{3.2}$$

It is further assumed that the functions f_k are given by explicit closed-form expressions. Black-box optimization, for instance, is not considered. To demand box bounds for the decision variables is not very restrictive. They are often dictated by the application or implied by the objective and other constraints. Equality constraints could be included in (3.2), but it is assumed that they are remodeled as two inequalities.

Most optimization algorithms will handle the linear constraints ($A\boldsymbol{x} \leq \boldsymbol{c}$, $B\boldsymbol{x} = \boldsymbol{d}$) separately. They are included among the general constraints here for brevity. It can also be useful to indicate in the input which functions are convex if it is known. Recognizing convexity is hard [5], except for special cases like quadratic functions. The convex programming solver CVX [33] recognizes some advanced convex structures by enforcing an input format where only convexity-preserving constructs are allowed. More general convex constraints like linear matrix inequalities [24] could be included in (3.2) if supported by the software. Some parameter calculation methods in Papers IV and V require the solution of semidefinite programming (SDP) subproblems.

The αBB methods described in Papers I–V are applicable to mixed-integer nonlinear programming (MINLP), but no special adaptations for discrete variables were made. Some methods require a certain degree of smoothness, twice continuous differentiability is sufficient for αBB.

Problems of type (3.2) may also appear as discretizations or subproblems in solving more general models, like semi-infinite programs, bilevel programs and certain problems from control theory or robust optimization.

3.3 Branch and Bound

The most general global algorithms for problems of type (3.2) all contain some element of branch-and-bound, a type of divide-and-conquer algorithm. Branch-and-bound methods were first used for mixed-integer linear programming (MILP) problems, pioneered by Land and Doig [69] and Dakin [34]. Falk and Soland adapted the concept to more general nonconvex problems [39].

Some notable global algorithms for nonlinear problems, like Generalized Benders Decomposition by Geoffrion [44], Outer Approximation by Duran and Grossmann [38], and the Extended Cutting Plane method by Westerlund and co-authors [126, 127], do not branch the domain of the decision variables. However, these algorithms require the problems to have certain convex substructures. Branching may, moreover, be used

implicitly by these algorithms as they pass on subproblems, commonly MILP problems, to some subsolver. MILP problems can be solved by pure cutting plane methods but these are usually inefficient compared to hybrid branch-and-cut methods [32, 130].

The idea of branch-and-bound in global optimization is the same as when used for MILP optimization. The domain of the decision variables is successively split into smaller subdomains (branching) and stored in a list of open nodes. Rigorous lower bounds are calculated for the nodes and the existence of feasible solutions is tested with local methods (bounding). If the lower bound indicates that the node cannot improve the best known solution, within a tolerance ε, the node is fathomed and removed from the list (pruning). The decreasing subdomains allow tighter tractable approximations of the problem, and the procedure will converge under some mild conditions, see Horst and Tuy [60].

The theoretical difference between branch-and-bound in MILP and (MI)NLP optimization is the type of convergence to the global optimum. The branching tree for an MILP problem is always finite and the global optimum can, in principle, be exactly determined. When branching on continuous variables the gap between lower and upper bounds might only converge to 0 in the limit [60]. An absolute error tolerance ε must be accepted in (MI)NLP methods.

The main steps of a typical branch-and-bound (MI)NLP method are presented below. Similar descriptions are given in many publications, *e.g.*, [40, 53, 60, 96, 103, 109], varying in detail and the type of descriptions (text, pseudo-code, mathematical notation). Figure 3.1 shows a flowchart of the same steps. This description was adapted from Ryoo and Sahinidis's article [103]. Nodes are described here as ordered pairs (D, L) containing a box domain and a rigorous lower bound for the problem on that domain.

1. **Initialization**
   Create a list $\mathcal{P}$ of open nodes and add the node $([\boldsymbol{x}^L, \boldsymbol{x}^U], -\infty)$ to the list. Set the best known global upper bound and solution (default values $U = +\infty$, x^{best} undefined). Goto 2.

2. **Node Selection**
   If $\mathcal{P}$ is empty, stop. Otherwise, apply a selection rule to choose an open node S and remove it from $\mathcal{P}$. Goto 3.

3. **Pre-processing**
   Apply bounds tightening to improve the variable bounds if possible. Goto 4.

4. **Lower Bounding**
   Calculate a rigorous lower bound S_L for the problem on the domain S_D. If $S_L > U - \varepsilon$, discard the node and Goto 2. Otherwise Goto 5

5. **Upper Bounding**
   Use local methods to find a feasible solution. If a solution was found that improves the current upper bound, update U and x^{best}, discard any node $T \in \mathcal{P}$ with $T_L > U - \varepsilon$. Goto 6.

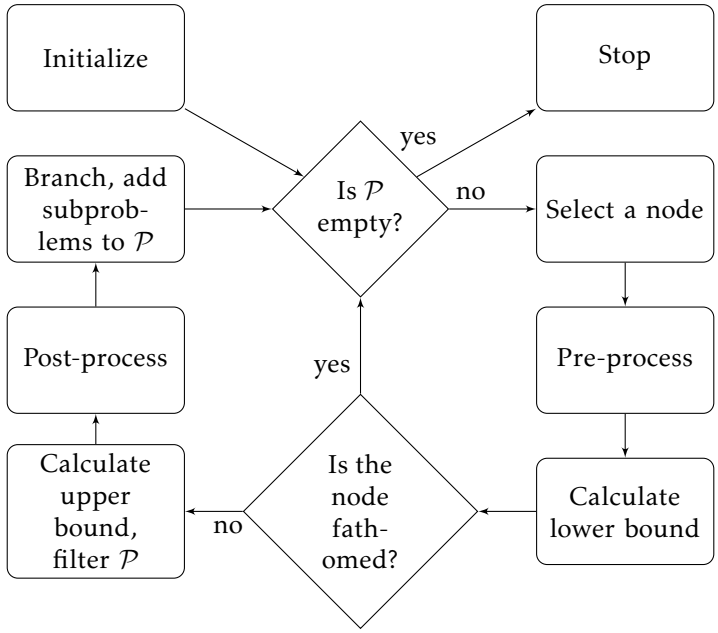

Figure 3.1: A flowchart of the major steps in a branch-and-bound algorithm.

6. **Post-processing**
   Apply bounds tightening to improve the variable bounds if possible. Goto 7.

7. **Branching**
   Apply a branching rule to generate subproblems from S. Add the subproblems to $\mathcal{P}$ and Goto 2.

In other descriptions the branch-and-bound algorithm will keep track of the smallest current lower bound L. Then the stopping criterion is to stop when $U - L < \varepsilon$. This is equivalent to exhausting all promising nodes and stopping when the list $\mathcal{P}$ is empty.

A best-first selection rule is commonly used for selecting the next node to study. In the lower bounding step the subproblem is replaced by a simpler one providing a lower bound, usually a convex or linear relaxation. The relaxed problem is easier to solve to global optimality and the solution provides a lower bound for the node. Many global solvers are designed in a modular way. Subproblems, such as the relaxed problem in the lower bounding step and the local search in the upper bounding step, are usually deferred to other solvers.

Pre- and post-processing may greatly improve the practical performance of an algorithm. The idea is to reduce the variable bounds in the node without discarding any potential optimum. Successful bounds tightening results in tighter approximations of the node in the lower bounding step. This in turn gives shorter search paths before the subdomains are fathomed. Bounds tightening is also known as bound reduction,

domain contraction, range reduction, etc. Three main varieties of bounds tightening methods have been described in the literature. These are based on feasibility and constraints propagation, on optimization of the variable bounds, and on reduced cost considerations [20, 113].

The branching rule is another important design choice in a global branch-and-bound algorithm. Simple rules like bisecting all variables or cycling through the variables will usually hamper the performance. Better strategies will estimate the effect of the branching on the problem approximation, especially for constraints that are violated by the relaxed solution. Branching rules for (MI)NLP are discussed in, *e.g.*, [1, 4, 114]. Both bounds tightening and the choice of branching variable are trade-offs between time and quality.

Integer or binary variables deserve special attention in the bounds tightening and branching steps. An integer-relaxed interval like $[0,1]$ collapses to two specific values $\{0,1\}$ when branched. The field of constraint programming has long dealt with discrete variables and constraints of logical or combinatorial nature [12]. Hooker [59] surveys the differences between optimization and constraint programming, and how the fields are converging. He also makes a case for exploiting the stronger modeling capabilities of constraint programming: *"it is inefficient to obliterate structure by modeling with an atomistic vocabulary of inequalities and equations and then try to rediscover the structure automatically."*

Terminology like *branch-and-reduce* is sometimes used to indicate that special steps are taken to reduce variable bounds (pre- and post-processing in the description above) [102]. *Branch-and-cut* implies the introduction of derived constraints in the nodes to tighten the approximations of the feasible domain, the concept is widely used in integer programming [82].

The fact that quadratic programming and many other optimization problems are NP-hard gives reason to believe that all general branch-and-bound algorithms have exponential (or at least superpolynomial) worst-case runtimes. This has been proven for some specific algorithms in integer programming, see [62], but it may be hard to formalize and prove more generally. Intuitively it can be attributed to the "curse of dimensionality", a term coined by Bellman [19]. A simple visualization is the function $\prod_{i=1}^{n} \sin(x_i)$ on some box domain, see Figure 3.2. The number of global minima grow exponentially with the number of variables. A branch-and-bound method would have to branch and refine the lower bounds around each minimum to establish the global optimum within ε-tolerance. The above example is separable and open to analytical solution on a box domain, but multiple minima of the same magnitude appear also for complicated models that preclude a simple analysis.

The convergence behavior of branch-and-bound methods to one isolated global minimum has been described by Kearfott and Du [37, 65], who called it the *cluster problem*. Let w describe the "width" of a box domain:

$$w([\boldsymbol{x}^{\boldsymbol{L}}, \boldsymbol{x}^{\boldsymbol{U}}]) := \max_{i=1,2,\ldots,n} (x_i^U - x_i^L).$$

The relaxations have convergence order γ if there exists a constant $K > 0$ such that the

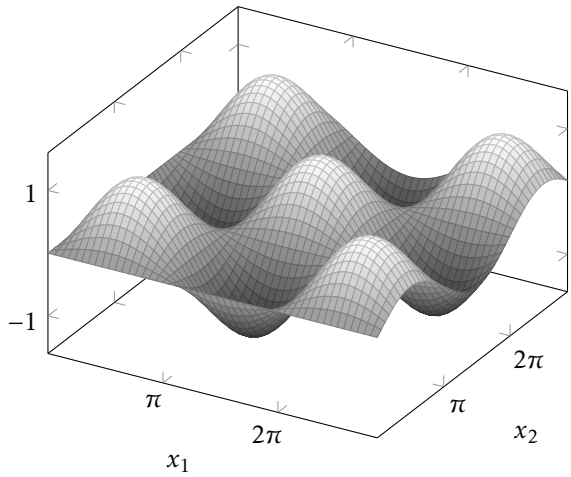

Figure 3.2: The graph of $\sin(x_1)\sin(x_2)$, $\boldsymbol{x} \in [0, 9]^2$, a simple example of a function with several global minima.

underestimation error is bounded by $Kw(D)^\gamma$ for all box domains D. Bompadre and Mitsos [23] and Wechsung et al. [122] derive convergence order results for the general underestimation methods described in Section 4.2. The central result on the cluster problem is that convergence order 2 is a limiting case, below 2 the number of studied nodes may depend exponentially on the reciprocal of the tolerance, $1/\varepsilon$.

The next chapter describes methods to relax and convexify individual constraints, through some expression-specific relaxations and two general methods. These procedures can be applied in the lower bounding step of a branch-and-bound algorithm.

Chapter 4

Convex Underestimation

Let $\check{f}_k$ denote a convex underestimator of the function f_k on the domain D. Replacing the objective and the constraint functions in (3.2) by convex underestimators is a form of convex relaxation. The solution to the relaxed problem provides a lower bound on the original problem:

$$\begin{array}{l} \min\{f_0(\boldsymbol{x}) : f_1(\boldsymbol{x}) \leq 0, \ldots, f_n(\boldsymbol{x}) \leq 0, \boldsymbol{x} \in D\} \quad \geq \\ \min\{\check{f}_0(\boldsymbol{x}) : \check{f}_1(\boldsymbol{x}) \leq 0, \ldots, \check{f}_n(\boldsymbol{x}) \leq 0, \boldsymbol{x} \in D\} \end{array} \tag{4.1}$$

Figure 4.1 illustrates the relaxation of a constraint function.

The tightest convex relaxation of the feasible set is the convex hull of the feasible set, as pointed out in [128]. This hull is usually hard to describe in some practical form. Some relaxation techniques handle many constraints simultaneously to achieve tighter approximations than the individual relaxations in (4.1), but this requires special structure in the problem. The Refomulation-Linearization Technique [107] adds cutting planes derived from two or more constraints to strengthen the relaxation. The semidefinite programming approximation algorithm for maximum cut problems [45] is in some sense a simultaneous relaxation of all the (binary) constraints. Liberti [71] gave a categorization of several reformulation and convexification techniques.

This chapter describes how individual functions can be relaxed to provide lower bounds as in (4.1). Section 4.1 lists some explicitly described convex and concave envelopes. Two general methods of constructing convex underestimators, αBB and factorable program relaxations, are described in Section 4.2. Convex relaxations and envelopes are interesting in their own right, but the focus here is on convenient representations for use in optimization.

4.1 Convex Envelopes

The *convex envelope* of the function f on the domain D is defined as the greatest, pointwise and on the whole domain, convex underestimator of f on D. Other characterizations are possible [112]. The *concave envelope* is defined analogously. Note that the

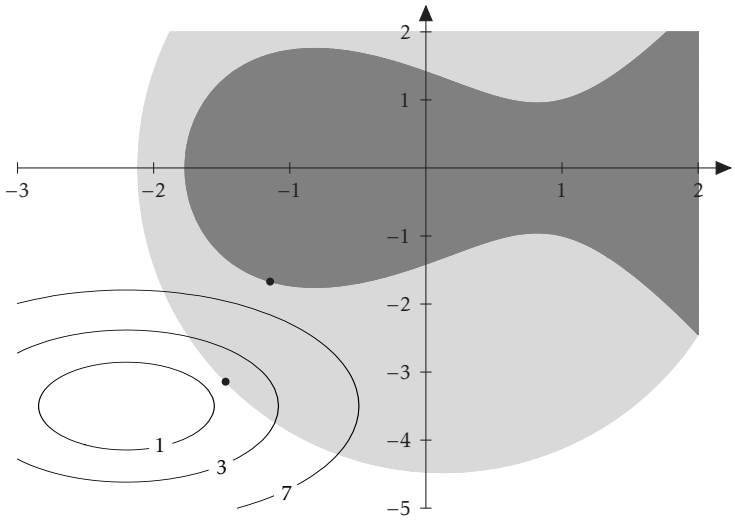

Figure 4.1: Minimization of a convex objective f (level curves) on $[-3,2]\times[-5,2]$ under the constraint $-x^3+2x+y^2-2\leq 0$ (dark gray area). A convex relaxation of the constraint is given by $4x^2-x+y^2-20\leq 0$ (light gray area). Note that the relaxed feasible set is larger than the convex hull. The global minimum, $f(-1.14,-1.67) = 10.71$, and the lower bound from the relaxation, $f(-1.47,-3.14) = 1.58$, are indicated.

envelopes depend on the domain and may change if the domain is restricted. I will denote convex and concave envelopes of f over D by $\mathrm{conv}_D f$ and $\mathrm{conc}_D f$, respectively.

The typical way to relax a model using envelopes is to substitute the functional expression $f(\boldsymbol{x})$ with a new variable y_f. The new variable is then constrained by

$$\mathrm{conv}_D f \leq y_f \leq \mathrm{conc}_D f, \tag{4.2}$$

or just one of the inequalities, depending on the constraint that is being relaxed.

Envelopes have been described for a plethora of functions, this section provides just a small sample of the existing descriptions. A collection of old and new results are found in a book by Tawarmalani and Sahinidis [113]. Khajavirad and Sahinidis [66] give many relevant references on convex hulls.

Envelopes for univariate convex and concave functions on D are extremely simple. Suppose f is a concave function of x_1 only. Then the concave envelope coincides with the function and the convex envelope is a linear interpolation between the endpoints:

$$\mathrm{conv}_D f = f(x_1^L) + \frac{f(x_1^U)-f(x_1^L)}{x_1^U-x_1^L}(x-x_1^L).$$

Let $D = [\boldsymbol{x^L},\boldsymbol{x^U}]$ in the rest of the section. The convex and concave envelopes of a

bilinear term x_1x_2 on D are:

$$\begin{aligned}\mathrm{conv}_D(x_1x_2) &= \max\left(x_1^L x_2 + x_1 x_2^L - x_1^L x_2^L,\, x_1^U x_2 + x_1 x_2^U - x_1^U x_2^U\right)\\ \mathrm{conc}_D(x_1x_2) &= \min\left(x_1^L x_2 + x_1 x_2^U - x_1^L x_2^U,\, x_1^U x_2 + x_1 x_2^L - x_1^U x_2^L\right).\end{aligned}$$

These envelopes appear in [77] and are often called McCormick envelopes. A simple proof is found in [9]. The max and min expressions are not everywhere differentiable. However, when used in a relaxation of type (4.2) they can be reformulated as linear constraints on a substitution variable $y_{x_1x_2}$:

$$\begin{aligned} y_{x_1x_2} &\geq x_1^L x_2 + x_1 x_2^L - x_1^L x_2^L\\ y_{x_1x_2} &\geq x_1^U x_2 + x_1 x_2^U - x_1^U x_2^U\\ y_{x_1x_2} &\leq x_1^L x_2 + x_1 x_2^U - x_1^L x_2^U\\ y_{x_1x_2} &\leq x_1^U x_2 + x_1 x_2^L - x_1^U x_2^L. \end{aligned} \tag{4.3}$$

The envelopes of a multilinear term $\prod_{i\in I} x_i$ on $[0,1]^n$ are [113]:

$$\begin{aligned}\mathrm{conv}_{[0,1]^n}\left(\prod_{i\in I} x_i\right) &= \max\left(0,\, \sum_{i\in I} x_i - |I| + 1\right)\\ \mathrm{conc}_{[0,1]^n}\left(\prod_{i\in I} x_i\right) &= \min\{x_i : i \in I\}.\end{aligned}$$

The expressions also give the tightest possible convex integer-relaxation of $\prod_{i\in I} x_i$ on $\{0,1\}^n$.

Any concave function on a convex (bounded) polytope has a piecewise linear convex envelope. The vertices of the envelope coincide with the vertices of the polytope and the envelope is said to be *vertex polyhedral*. Tardella showed that the convex envelope is vertex polyhedral for the larger class of *edge-concave* functions [112]. A function with a polytope domain is edge-concave if it is concave along all line segments that are parallel to an edge of the polytope. This includes multilinear functions like $x_1x_2x_3 - x_3x_4$ on D.

Meyer and Floudas give an algorithm to find a triangulation (into simplices) matching the convex envelope for an edge-concave function in three dimensions [79]. The possible number of facets grows quickly with the dimension and determining the complete convex envelope is intractable in high dimensions.

Tawarmalani and Sahinidis discuss the envelopes and other relaxations of the fraction $f(x,y) = x/y$ [113]. The convex envelope on $E = [x^L, x^U] \times (0,\infty)$ with $x^L > 0$ has a compact description:

$$\mathrm{conv}_E\left(\frac{x}{y}\right) = \frac{1}{y}\left(\frac{x + \sqrt{x^L x^U}}{\sqrt{x^L} + \sqrt{x^U}}\right)^2.$$

The concave envelope on $F = [x^L, x^U] \times [y^L, y^U]$ with $x^L > 0$, $y^L > 0$ is polyhedral:

$$\mathrm{conc}_F\left(\frac{x}{y}\right) = \frac{1}{y^L y^U}\min\left(xy^U - x^L y + x^L y^L,\, xy^L - x^U y + x^U y^U\right).$$

An alternative relaxation of x/y on F is to substitute the fraction with a new variable $z_{x/y} \in \left[\frac{x^L}{y^U}, \frac{x^U}{y^L}\right]$, and then relax the relation $x = yz_{x/y}$ with the McCormick envelopes for bilinear terms.

4.2 Two General Methods

This section outlines two methods for constructing convex underestimators (and concave overestimators) for generic function expressions, αBB and factorable program relaxation.

The αBB method adds certain perturbations to a nonconvex function such that the perturbed functions satisfies second order convexity conditions. Since second order derivatives are needed, the method applies to twice continuously differentiable functions, $\mathcal{C}^2$. Zlobec [131, 132] shows how αBB can in principle be extended to a slightly larger class of functions than $\mathcal{C}^2$, including all functions with Lipschitz-continuous gradient. Most of the αBB literature covers underestimation, analogous results on concave overestimators follow from applying the methods to $-f$.

Let H_f denote the Hessian of f (a function of $\boldsymbol{x}$) and H_P the (constant) Hessian of the perturbation. Define the relation $A \succeq B$ on symmetric matrices to mean that $A - B$ is positive semidefinite. Convexifying perturbations satisfy:

$$\forall \boldsymbol{x} \in D: \; H_f(\boldsymbol{x}) + H_P \succeq 0. \tag{4.4}$$

Several sufficient conditions have been derived from (4.4), see [3], Paper II and Paper V.

The diagonal elements of H_P are $2\alpha_i \geq 0$, the factor 2 is included by convention, and the off-diagonal elements are denoted β_{ij}. The perturbation corresponding to α_i is

$$-\alpha_i(x_i^U - x_i)(x_i - x_i^L). \tag{4.5}$$

If a "nondiagonal" αBB method is used the elements β_{ij} correspond to perturbations of the form:

$$\beta_{ij}x_ix_j + |\beta_{ij}| \cdot \begin{cases} \max\begin{pmatrix} -x_ix_j^L - x_i^Ux_j + x_i^Ux_j^L, \\ -x_ix_j^U - x_i^Lx_j + x_i^Lx_j^U \end{pmatrix}, \text{ if } \beta_{ij} > 0 \\ \max\begin{pmatrix} x_ix_j^L + x_i^Lx_j - x_i^Lx_j^L, \\ x_ix_j^U + x_i^Ux_j - x_i^Ux_j^U \end{pmatrix}, \text{ if } \beta_{ij} < 0. \end{cases} \tag{4.6}$$

Naturally $\beta_{ij} = \beta_{ji}$, and by convention the perturbation is added for elements above the diagonal, $j > i$, instead of half-and-half. The max() expressions are related to the McCormick envelopes (4.3), they are the negative of the concave envelope of $\text{sgn}(\beta_{ij})x_ix_j$. Paper II was the first extensive discussion of nondiagonal αBB. There it is argued that (4.6) is the natural and optimal realization of a β perturbation. The expressions can be modeled linearly with auxiliary variables and constraints. All perturbation terms are nonpositive which ensures the underestimation property.

The maximum approximation error is linearly proportional to α_i and $|\beta ij|$:

$$\max_{\boldsymbol{x} \in D} f(\boldsymbol{x}) - \check{f}(\boldsymbol{x}) \; = \; \sum_i \alpha_i \frac{(x_i^U - x_i^L)^2}{4} + \sum_i \sum_{j>i} |\beta_{ij}| \frac{(x_i^U - x_i^L)(x_j^U - x_j^L)}{4}.$$

This gives a criterion for choosing among the perturbations satisfying (4.4). It is shown in Paper IV that the average error, an L^1 norm, is also linear in α_i and $|\beta_{ij}|$, with a relatively lighter weight on $|\beta_{ij}|$. The L^1 norm was then used alongside maximum errors for the test runs in Paper V.

The parameter calculations should not be too costly. The *diagonal scaled Gerschgorin method* calculates α_i as an explicit formula and gives good results compared to some more complicated methods [3]. The methods developed in Papers II and V solve linear or convex programs to determine perturbation parameters. The approximation errors of different methods can be compared empirically and, to some extent, theoretically.

The term αBB, with 'BB' standing for Branch-and-Bound, has come to denote both the complete optimization algorithm described by Adjiman et al. [2, 3] and Floudas [40], and the method to calculate convex relaxations. I mostly refer to the latter.

The other approach, *factorable program relaxation*, is based on partitioning a functional expression into its constituent parts. An expression can be represented as a directed tree structure where the leaf nodes are constants or variables, and the other vertices denote a function from a predefined set of "elementary operations". The allowed operations are usually a superset of $\{+, -, \times, /$, negation, powers, exponentiation, logarithms$\}$. The method applies to a large part of (MI)NLP models in applications. It can also handle some nondifferentiable expressions.

The basic relations described by the internal nodes can be relaxed individually, usually introducing new variables to represent the relaxation like in (4.3). The result is a *lifting*, a reformulation with auxiliary variables. The original problem can be rewritten in "factorable" form as an intermediate step before relaxation. In a factorable program the constraints are simple applications of the elementary operations. For example, $x_1\sqrt{x_2} - 1 \leq 0$ can be replaced by:

$$\begin{cases} y_1 = \sqrt{x_2} \\ y_2 = x_1 y_1 \\ y_2 \leq 1. \end{cases}$$

McCormick described such a relaxation procedure in 1976 [77]. Some authors refer to the whole method as McCormick relaxation [83, 116]. Implementations with bounds tightening techniques and appropriate branching rules appeared in the 90's, *e.g.*, [103, 108].

If the same subexpression appears many times in a function (or in the full problem), it would be inefficient to introduce new relaxation variables for every occurrence. Instead the expressions can be represented as a directed acyclic graph (DAG). Figure 4.2 shows an expression represented as a tree and as a DAG.

Factorable program relaxations are easy to interpret as overestimation of the feasible set, but one can also demonstrate underlying convex underestimators (and concave overestimators). Let g (univariate) and h be functions from the set of elementary oper-

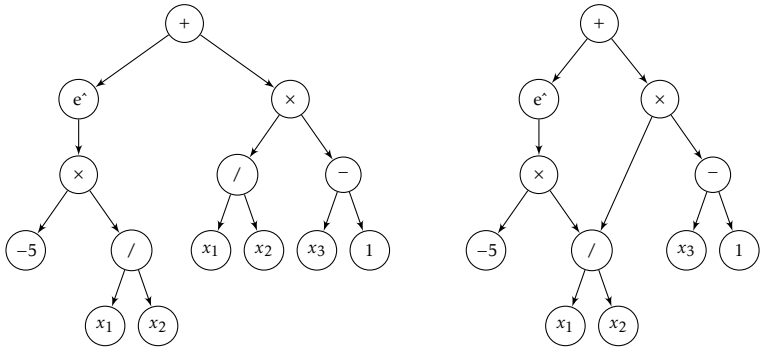

Figure 4.2: A tree and a directed acyclic graph representing the same expression, $\exp(-5\,x_1/x_2) + (x_1/x_2)(x_3 - 1)$. The second term can be written in different ways. The partition made here allows the same auxiliary variable and relaxation of x_1/x_2 to be used for the left and the right term.

ations, and $f := g \circ h$. The constraint $f(\boldsymbol{x}) \leq 0$ is reformulated as:

$$\begin{cases} y_1 = h(\boldsymbol{x}) \\ y_2 = g(y_1) \\ y_2 \leq 0, \end{cases}$$

where y_2 comes to represent the expression $f(\boldsymbol{x})$. The two equalities are relaxed as (the hat $\hat{}$ denotes a concave overestimator):

$$\begin{cases} \check{h}(\boldsymbol{x}) \leq y_1 \leq \hat{h}(\boldsymbol{x}) \\ \check{g}(y_1) \leq y_2 \leq \hat{g}(y_1) \end{cases} \tag{4.7}$$

The functions

$$\begin{aligned} \check{f}(\boldsymbol{x}) &= \min\{\check{g}(z) : \check{h}(\boldsymbol{x}) \leq z \leq \hat{h}(\boldsymbol{x})\} \\ \hat{f}(\boldsymbol{x}) &= \max\{\hat{g}(z) : \check{h}(\boldsymbol{x}) \leq z \leq \hat{h}(\boldsymbol{x})\} \end{aligned} \tag{4.8}$$

are a convex underestimator and a concave overestimator of f, respectively [77, 116]. Eliminating y_1 from (4.7) gives:

$$\check{f}(\boldsymbol{x}) \leq y_2 \leq \hat{f}(\boldsymbol{x})$$

and the implicit under- and overestimators become clear. Note that some explicit numerical bounds on y_1 may be needed to construct $\check{g}$ and $\hat{g}$. When h is an elementary function the exact bounds can usually be inferred from $\boldsymbol{x} \in D$. Otherwise overestimating bounds can be calculated by interval arithmetic [87].

One should not confuse these two general approaches with particular implementations. The specific implementations use known convex and concave envelopes for some functions. In addition other specialized relaxations may be used, different from the general approach. It is reasonable to include more specialized underestimators as efficient

descriptions are discovered. Paper IV is a comparison between αBB and an impressive but computationally heavy method for underestimating polynomials.

No off-the-shelf solver seems to rely heavily on αBB methods. Gatzke et al. [43] described the hybrid use of αBB and factorable program relaxations. Development of the related Fortran library DAEPACK [17] has ceased. The early αBB descriptions [2, 3, 10, 40] suggest using convex envelopes for bilinear and univariate concave terms, and other tight relaxations for trilinear, fractional and "fractional trilinear" terms ($xyz, x/y, xy/z$).

The factorable decompositions are found in several global solvers, *e.g.*, BARON [113], Couenne [20] and ANTIGONE [80]. BARON has arguably been the most succesful global solver for some time. Besides the comparison made by Neumaier et al. [89] in 2004, newer benchmark results are given on the BARON and ANTIGONE websites. BARON switches to specialized solver modules for some structured problems, *e.g.*, quadratic programming and linear multiplicative programming. Couenne is an active open-source project within the COIN-OR framework [72], making it possible for developers to contribute to the project or to specialize the code for their own applications. ANTIGONE builds on the framework used for GloMIQO, a quadratic optimization (MIQCQP) solver developed by Misener and Floudas [81]. Both solvers utilize convex (concave) envelopes for edge-concave (edge-convex) expressions of up to four variables.

The three mentioned solvers all relax the convexified model further by linearizing it. This gives looser approximations, but the solvers benefit from the speed and robustness of linear programming solvers. Tawarmalani and Sahinidis [113] discuss how to choose linearization points for the univariate components. This approximation process is called a sandwich algorithm. Several rules for placing p linearization points make the largest vertical distance decrease as $O(1/p^2)$ [101, 113].

A simple one-dimensional example, $f(x) = e^{-x^2}$, $x \in [0, 2]$, serves to illustrate some points. The under- and overestimators are plotted in Figure 4.3. The relaxation errors increase further if the under- and overestimators are linearized.

The αBB perturbations are found by studying the second derivative $f''(x) = e^{-x^2}(-2+4x^2)$. A basic interval extension yields the interval approximation:

$$f''([0,2]) \subseteq e^{-[0,2]^2}(-2 + 4 \cdot [0,2]^2) = [e^{-4}, 1] \cdot [-2, 14] = [-2, 14].$$

The lower bound is exact, $f(0) = -2$, but the upper bound is loose. The actual upper bound, $4e^{-3/2} \approx 0.893$, can be found symbolically at a zero of the third derivative. It is of course hard in general to determine exact interval bounds. If the second derivative is given without simplification as $f''(x) = -2e^{-x^2} + 4e^{-x^2}x^2$, then the basic interval extension is looser, $f''([0,2]) \subset [-2, 16 - 2e^{-4}]$.

All methods for determining the convexifying parameter ($\breve{\alpha} \geq 0$) coincide in the one-dimensional case. The underestimator is convex if and only if:

$$\min_{x \in [0,2]} f''(x) + 2\breve{\alpha} \geq 0.$$

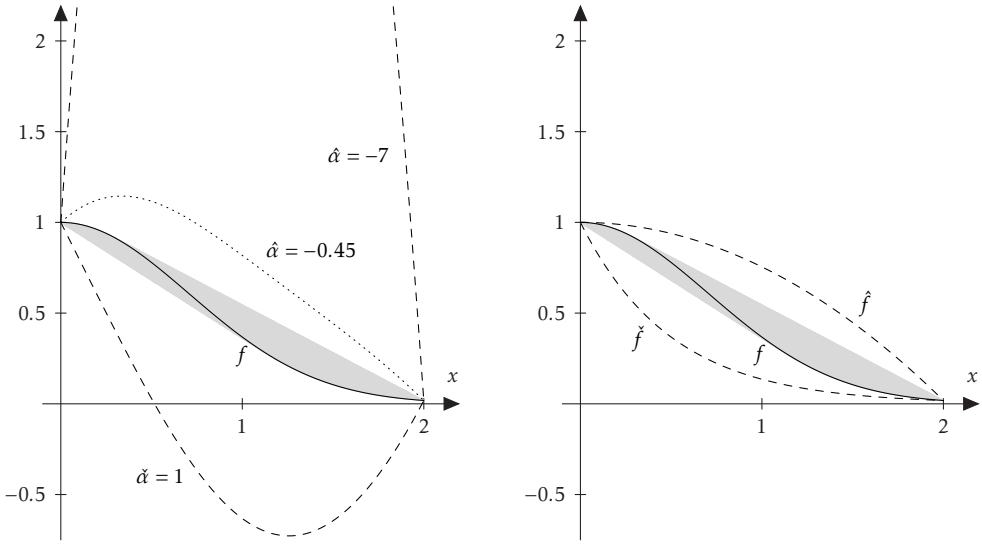

Figure 4.3: The function $f(x) = e^{-x^2}$ (solid) together with αBB relaxations (left, dashed) and factorable program relaxations (right, dashed). A better interval approximation of $f''([0,2])$ shows that the parameter $\hat{\alpha} = -0.45$ is sufficient, giving a tighter overestimator (dotted). The gray area is the convex hull of the graph.

The smallest possible parameter is:

$$\check{\alpha} = -\frac{1}{2} \min_{x \in [0,2]} f''(x) = 1.$$

The same perturbation (4.5) is used for the concave overestimator, with a parameter $\hat{\alpha} \leq 0$. The overestimator is concave if:

$$\max_{x \in [0,2]} f''(x) + 2\hat{\alpha} \leq 0.$$

The interval approximation $f''([0,2]) \subset [-2,14]$ gives $\hat{\alpha} = -7$, but the best possible parameter, with the lowest absolute value, is $\hat{\alpha} = -2e^{-3/2} \approx -0.45$. This shows how approximation errors of the intervals can affect the tightness of the relaxations. Both values of $\hat{\alpha}$ are illustrated in Figure 4.3.

To construct the factorable relaxations the function is first reformulated with auxiliary variables:

$$\begin{cases} y_1 = -x^2 \\ y_2 = e^{y_1}. \end{cases}$$

The right-hand side expressions are concave and convex, respectively. Relaxing the

elementary operations gives:

$$\begin{cases} -2x \leq y_1 \leq -x^2 \\ -4 \leq y_1 \leq 0 \\ e^{y_1} \leq y_2 \leq e^{-4} + \frac{1-e^{-4}}{4}(y_1 - (-4)) \\ e^{-4} \leq y_2 \leq 1. \end{cases}$$

Monotonicity makes it easy to eliminate y_1 and get explicit expressions for (4.8):

$$\begin{aligned} \check{f}(\boldsymbol{x}) &= e^{-2x} \\ \hat{f}(\boldsymbol{x}) &= e^{-4} + \tfrac{1-e^{-4}}{4}\left(-x^2 - (-4)\right). \end{aligned}$$

The envelopes of this specific example are not hard to work out. They are piecewise defined by the function itself and one of its tangents, and the breakpoint can be determined up to numerical precision. The factorable program relaxations are clearly not the envelopes in this case, but they are fairly tight for a general method.

The improved αBB overestimator ($\hat{\alpha} = -0.45$) is slightly tighter than $\hat{f}$ towards the right end of the interval, neither method dominates strictly. However, the αBB relaxation is definitely looser on average. This may be the case for many realistic examples, although no definitive comparison of the two methods has been published.

The convergence rate analysis by Bompadre and Mitsos [23] is one interesting approach. Recall the concept of convergence order from Chapter 3. A "relaxation scheme" for an expression has convergence order γ if the approximation error $\hat{f}(\boldsymbol{x}) - \check{f}(\boldsymbol{x})$ (or a one-sided error for inequality relaxations) is bounded by $Kw(D)^\gamma$ for some constant $K > 0$ and all box domains D. Bompadre and Mitsos show that αBB has (pointwise) convergence order 2 and prove several theorems on "McCormick relaxations". These results imply that many factorable program relaxations also converge quadratically.

Convergence order is thus not enough to choose one of the methods over the other. Perhaps bounds on the constant K could shed more light on differences between the methods. For αBB the constant can be chosen as a simple function of the parameters, $K = (\check{\alpha} - \hat{\alpha})/4$ in one dimension. To determine a sufficient K for a factorable program relaxation is likely to be more involved and dependent on the component relaxations. This is a possible topic for future research.

Chapter 5

Notes

This chapter contains some notes on Papers I–V, included in this thesis. The novel results and the connections to other research in the area are highlighted. The introduction sections in the papers themselves list additional relevant references.

5.1 Paper I

In 2010, Andreas Lundell and Prof. Tapio Westerlund were extending the Signomial Global Optimization (SGO) algorithm to handle nonconvexities besides signomials. The algorithm convexifies signomial terms with power and exponential transformations. It is often the case that different occurrences of a variable require different transformations. The relation to the original variables are then included and approximated by piecewise linear functions in such a way that the feasible area is overestimated [74, 125]. Lundell's PhD thesis describes how the choice of transformations can be optimized as an MILP problem, weighing such considerations as the number of original variables that are transformed and the number of different transformations [73].

The SGO algorithm does not explicitly do spatial branching. Instead the overestimated feasible set is tightened in each step by refining the approximations with more discrete variables. We were able to show how αBB type underestimators can be included and refined in much the same way as the signomial transformations. A function f on $[\boldsymbol{x}^{\boldsymbol{L}}, \boldsymbol{x}^{\boldsymbol{U}}] \subset \mathbb{R}^n$ is underestimated by:

$$f(\boldsymbol{x}) - \sum_{i=1}^{n} \alpha_i (x_i^U - x_i)(x_i - x_i^L) + \sum_{i=1}^{n} W_i.$$

The sum in the middle is the standard αBB perturbation. W_i is a piecewise linear approximation of $\alpha_i(x_i^U - x_i)(x_i - x_i^L)$, realized with some special ordered set representation (type 1 or type 2). Linear and constant parts can be shifted to W_i, leaving terms of the form $\alpha_i x_i^2$ in the left sum. Figure 5.1 shows a univariate function, its αBB underestimator and a piecewise refined underestimator.

I suggested that the work on spline underestimators by Meyer and Floudas [78] could be used in the SGO context. Michael Bussieck and Lutz Westermann at GAMS

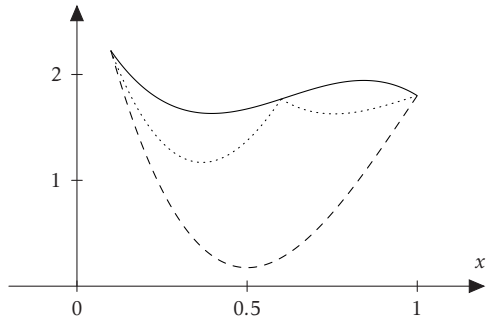

Figure 5.1: A smooth function with αBB underestimator (dashed) and a piecewise refined underestimator (dotted).

Development quickly granted our wish for stronger support of polynomial splines (and piecewise defined polynomials in general) in the GAMS modeling language. The extension of the SGO algorithm and details of implementation are described in Lundell et al. [75].

5.2 Paper II

Prof. Christodoulos A. Floudas gave two presentations at the OSE group's seminar in November 2010. He mentioned in passing the possibility of bilinear "nondiagonal" αBB perturbations. In March 2011 I had the opportunity to visit Prof. Floudas's research group, the Computer-Aided Systems Laboratory at Princeton University. Together with Ruth Misener, then PhD student, we developed expressions for the perturbations and a method to calculate the underestimation parameters (α, β). Akrotirianakis et al. published a conference paper on the subject in 2004 [8]. Our contribution presents a self-contained mathematical theory for the underestimators and makes the case that convexity and underestimation dictates a natural form for the β perturbations. The perturbations are essentially the same as in [8], but the expressions are more streamlined without the artificial distinction between positive and negative β values. Two special cases are described where optimal β parameters can be determined without solving a linear program. The nondiagonal underestimators are compared with diagonal αBB in terms of root node relaxations for a set of test functions.

5.3 Paper III

In this conference paper the central results from Paper II are summarized and discussed. A small branch-and-bound solver was implemented in C++ to make node count comparisons between diagonal and nondiagonal αBB. The lower bounding was performed with cutting planes and calls to the Gurobi solver [50]. Upper bounds were found with the local solver CONOPT [36]. The algorithm included feasibility-based

bound reduction [40]. The node counts favor nondiagonal αBB as expected, but a more full-fledged and integrated implementation would be needed to compare solution times of the methods.

5.4 Paper IV

Lasserre and Thanh's article [70] was my introduction to sum-of-squares optimization. Sum-of-squares polynomials with degrees less than or equal to some fixed numbers can be modeled by semidefinite programming constraints. This provides some interesting modeling capabilities, see for example the PhD thesis by Parrilo [93].

The usefulness of sum-of-squares polynomials is further enhanced by results from algebraic geometry. A semi-algebraic set is a subset of Euclidian space defined by polynomial inequalities, $K := \{\boldsymbol{x} \in \mathbb{R}^n : p_1(\boldsymbol{x}) \geq 0, \ldots, p_m(\boldsymbol{x}) \geq 0\}$. The polynomials p_i could for example describe a box domain: $p_i(\boldsymbol{x}) := (x_i^U - x_i)(x_i - x_i^L) \geq 0$. Putinar's Positivstellensatz [95] from 1993 gives a characterization of positive polynomials on K, under some light assumptions that are satisfied in our application. Lasserre and Thanh use Putinar's Positivstellensatz to model both underestimation and convexity in calculating tight convex underestimators of polynomials. The degrees of the participating polynomials are limited to make the problem finite-dimensional and turn it into a semidefinite program. The polynomial coefficients are optimized by minimizing an L^1 error objective.

Lasserre and Thanh's (L&T) method is summarized in the paper, and the model size is related to the number of variables and the degrees of the polynomials. The method was implemented with CVX [33] and Matlab. The results are like the theory predicts; the tightness of the produced underestimators is impressive but the calculation effort grows quickly with the polynomial degrees. With a modest number of variables the practical degree limits may range from 3 to 5. Note that the convexity of the underestimators is proven by advanced methods, and they would not be recognized by CVX's "structured convex programming" paradigm without some extensions.

The L^1 error objective for αBB underestimators was derived to compare αBB with the L&T method. The influence of a parameter α_i on the averaged L^1 error is linear and described by the coefficient:

$$\frac{\int_{[\boldsymbol{x}^L, \boldsymbol{x}^U]} (x_i^U - x_i)(x_i - x_i^L)\,\mathrm{d}\boldsymbol{x}}{\int_{[\boldsymbol{x}^L, \boldsymbol{x}^U]} \mathrm{d}\boldsymbol{x}} = \frac{1}{6}(x_i^U - x_i^L)^2.$$

Note that the square of the interval width appears as for the maximum error, only with a different factor. This means that diagonal αBB variants will optimize both objectives simultaneously. The influence of $|\beta_{ij}|$ is calculated analogously by integrating the piecewise defined parts of the nondiagonal perturbation. The expression was integrated using symbolical calculation software. The different cases of positive and negative β_{ij} give the same coefficient:

$$\frac{1}{12}(x_i^U - x_i^L)(x_j^U - x_j^L).$$

Again, the expression differs only by a factor from the maximum error objective. The relative weight of $|\beta_{ij}|$ is smaller than for maximum errors, increasing the incidence of off-diagonal terms in optimal perturbations.

The quadratic case is stated in a more explicit form in the paper. The L&T underestimator in this case corresponds to the perturbation:

$$-\sum_{i=1}^{n} \sigma_i (x_i^U - x_i)(x_i - x_i^L) - [\boldsymbol{x}'\ 1]\, C \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}.$$

The left sum is equivalent to diagonal α perturbations, so the method is more general than diagonal αBB.

In the quadratic case the Hessian is a constant matrix and it is possible to model positive semidefiniteness exactly as a semidefinite programming constraint:

$$\nabla^2 f(\boldsymbol{x}) + \begin{bmatrix} 2\alpha_1 & \beta_{1,2} & \cdots & \beta_{1,n} \\ \beta_{1,2} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \beta_{n-1,n} \\ \beta_{1,n} & \cdots & \beta_{n-1,n} & 2\alpha_n \end{bmatrix} \geq 0.$$

The αBB is different from purely polynomial underestimators because of the piecewise linear term in the perturbation. It gave an equal or better L^1 error on all instances of a randomly generated test suite and a tighter lower bound in a majority of the cases.

The two methods have different advantages. The L&T method applies to polynomials and gives tight underestimators, if one is willing to devote the computational resources to the underestimation subalgorithm. The αBB method is a general-purpose underestimation algorithm which applies to any sufficiently smooth function. It is at a disadvantage on polynomials, because Hessian interval approximations can be quite broad for polynomial degrees ≥ 3.

5.5 Paper V

The tightness of αBB-type underestimators depends on the width of the interval approximations and the sufficient conditions used to ensure convexity. The interval estimates can sometimes be improved by symbolic manipulation of expressions or more advanced interval methods, see Neumaier [87]. Even when the intervals are exact, by luck or through optimization, they may be wide. Since the set of occurring Hessian values is overestimated, the sufficient conditions for convexity are conservative.

Another reason for oversized perturbations is that the problem of bounding the eigenvalues of an interval matrix is NP-hard, even in the symmetric case [100]. The methods based on Gerschgorin circles, for example, give exact bounds when the interval matrix is diagonal but are conservative in general.

Paper V presents new alternatives for calculating αBB parameters. It was inspired by an eigenvalue theorem in linear algebra, Brauer's ovals of Cassini [27], and a theorem by Rohn dealing specifically with interval matrices [99]. Brauer's ovals give stronger

$$\begin{bmatrix} -3 & 1 & 1 \\ -2 & 1+3\imath & 1 \\ 1 & 1 & 3-\imath \end{bmatrix}$$

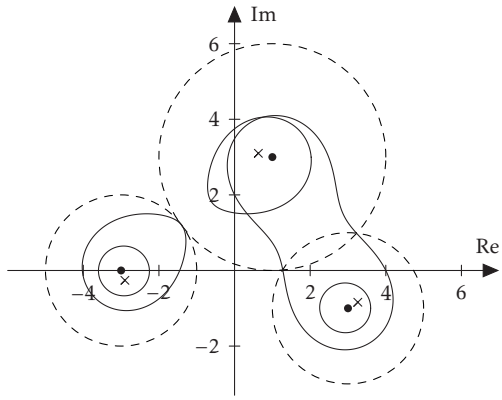

Figure 5.2: Gerschgorin's circles (dashed) and Brauer's ovals (solid) for a $\mathbb{C}^{3\times3}$ matrix. The midpoints of the circles (dots) and the actual eigenvalues (crosses) are indicated.

bounds on eigenvalues than Gerschgorin's circles in the sense that they use the same compounded information about the matrix and form a subset of the union of circles [117]. Figure 5.2 shows the circles and ovals for a given matrix. The example is complex-valued for the purpose of visualization, the eigenvalues of Hessian matrices are always real.

For symmetric instances of an interval matrix H', Rohn's eigenvalue bounds are simply

$$\lambda_{\min}(\mathrm{mid}(H')) - \rho(\mathrm{rad}(H')) \leq \lambda \leq \lambda_{\max}(\mathrm{mid}(H')) + \rho(\mathrm{rad}(H')),$$

where $\rho()$ denotes the spectral radius. Hladík et al. describe algorithms for calculating bounds on the ordered eigenvalues, $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$, of an interval matrix [57, 58]. Note that such bounds do not automatically translate to αBB parameters other than the homogenous alternative, $\alpha_i = \max(0, -\underline{\lambda_n})$ for all i, which is usually not competitive. The theorems of Gerschgorin and Brauer are different in that they associate eigenvalue bounds to each row/column of a matrix.

The theory mentioned here and reviewed in the paper is applied as convexity conditions in the search for αBB parameters. The resulting methods range in complexity from linear programs to semidefinite programs of modest size. The established diagonal Gerschgorin method, see Adjiman et al. [3], is included for comparison.

The methods are compared on a set of test functions and a large set of randomly generated interval matrices and relative performance curves are plotted. Scaling is applicable to all the methods described in the paper, and they are tested without scaling and with the intuitive interval width scaling. The L^1 error derived in Paper IV is used along with the maximum error. The diagonal Gerschgorin method is the fastest as it calculates parameters by evaluating simple formulas. To make comparisons more fair we include a version of this method which runs many different scalings and chooses the best set of parameters found, as suggested by one of the anonymous referees. Hladík has

recently derived interesting results on the diagonal Gerschgorin method, for example that interval width scaling is optimal in an important special case [56].

The results in the paper confirm that the diagonal Gerschgorin method is a good general-purpose method to calculate αBB parameters. One of the other methods may do better when there is some special structure in the Hessian, like sparsity or constant elements. This is discussed in Section 3 of the paper.

5.6 Contributions of the Author

Paper I I helped to clarify the role of the parameters when αBB underestimators are used in the SGO framework. Beyond that the paper is based on the work of the co-authors.

Paper II I found a continuous realization of the nondiagonal β parameters, and methods to choose the parameters while guaranteeing convexity. The realization and the calculation methods were further improved together with the co-authors. I worked out the theoretical results stated as theorems and the root node comparisons. Ruth Misener made the comparison with Akrotirianakis earlier work [8] and the in-depth example with the six-hump camel back function.

Paper III The theory in the paper is a summary of the central results in Paper II. I implemented a simple branch-and-bound solver, using the diagonal and nondiagonal αBB underestimators, to compare node counts.

Paper IV For this paper I implemented Lasserre and Thanh's method and made comparisons with αBB underestimation. I wrote the semidefinite program in the L&T method in a more explicit form for quadratic functions and derived an average error expression for αBB. Average error objectives were used in parallel with maximum errors in Paper V.

Paper V In this article I applied matrix theory results to get new sufficient convexity conditions for αBB parameters. I wrote the paper as a relatively self-contained description of the most promising parameter calculation methods, with and without scaling. I implemented all methods and compared root node errors on test functions and a large suite of randomly generated interval Hessians.

Chapter 6

Conclusions

The aim of my research has been to gain understanding of the strengths and weaknesses of αBB underestimation. The diagonal and the more recent nondiagonal variant, together with several parameter methods, have been studied and the results communicated in Papers I–V. It is often hard to put the most relevant questions in a form that allows them to be answered by a mathematically rigorous argument. The results are therefore a mixture of exact theorems and more interpretable comparisons on test functions. I feel satisfied with some of the questions that were answered exactly. Theorems 5.2 and 5.3 in Paper II explain something about how nondiagonal perturbations work and when they give tighter underestimators. Theorem 8 in Paper V shows how one parameter method dominates for underestimating quadratic functions, it is therefore a strong candidate for functions with nearly constant Hessians.

The large comparison on randomly generated interval matrices in Paper V shows that the simple idea behind diagonal Gerschgorin, one of the methods proposed in 1998 by Adjiman et al. [3], holds up very well against other proposed αBB parameter methods. The relative performance combined with the speed and simplicity of the method, a formula for the parameters, makes scaled diagonal Gerschgorin a general-purpose choice. Other methods can be considered if some beneficial structure of the Hessian is present. Hladík has recently made very interesting contributions on the choice of scaling for the diagonal Gerschgorin method [56]. He showed that scaling directly proportional to the interval width is optimal in an important special case, and how to heuristically improve the scaling otherwise. These results taken together represent the best available knowledge about how to implement αBB parameter methods.

The closest alternative to αBB methods, factorable program relaxations, have distinguished themselves by being the method of choice in some actively developed global solvers, see Section 4.2. The αBB methods do not appear in any equally successful and maintained solver software.

Important future research would be to make a full-fledged implementation of αBB and tune it with respect to design choices like the type of subproblems, the amount of bounds tightening, and the choice of branching rule. This would allow more conclusive practical comparisons between different αBB methods and against solvers relying on

factorable relaxations. A convenient platform to start from could be an existing open-source framework like COIN-OR [72].

The possibility that αBB underestimation is more effective for some expressions should not be ruled out. A conceivable situation where αBB may be at an advantage is expressions consisting of multiple layers of function compositions, $g = g_k \circ g_{k-1} \circ \ldots \circ g_0$. The convergence order of factorable program relaxations in such situations is discussed by Bompadre and Mitsos [23]. The size of αBB parameters will also deteriorate if simple interval arithmetic is used on the large analytical Hessian expressions that follow from the chain rule of differentiation. However, if all but the innermost function g_0 are one-dimensional functions $\mathbb{R} \to \mathbb{R}$, then interval methods and sampling techniques can give good bounds on the second derivative of the outer "factor" $g_k \circ g_{k-1} \circ \ldots \circ g_1$. Here lies an opportunity for hybrid methods.

The αBB literature is mostly restricted to quadratic perturbations. The extension to bilinear terms, and piecewise linear corrections, was natural since the elements of the perturbation Hessian stay constant. Perturbations parametrized in other ways are of course possible. Compare with the example in Figure 4.3, $f(x) = e^{-x^2}$, $x \in [0, 2]$. Only in the left part of the interval is the second derivative of the perturbation constrained to be positive. To the right, the perturbation could be nearly linear or even concave if it allows a tighter underestimator. As a simple example, third-degree polynomials in the individual variables, $a_i x_i^3 + b_i x_i^2 + c_i x_i + d_i$ form a superset of the diagonal αBB underestimators parametrized by α_i. If the parameters (a, b, c, d) are optimized, subject to keeping the perturbed function convex and underestimated, the result is at least as good as diagonal αBB.

The challenge is that a nonconstant perturbation Hessian makes it hard to give practical conditions for convexity. One cannot simply make an interval approximation of the perturbation, then the calculations trace the worst-case for the perturbation and nothing has been gained. Akrotirianakis and Floudas [6, 7] suggested perturbation terms with the exponential expression $-(1 - e^{\gamma_i(x_i - x_i^L)})(1 - e^{\gamma_i(x_i^U - x_i)})$. To calculate sufficient parameters γ_i required an iterative procedure, and in the end the performance was not better than αBB on box-constrained test problems [7].

An efficient underestimation method should also describe how the underestimation parameters can be updated and reused on a subdomain. In the case of αBB, once the parameters are calculated they are sufficient on any subdomain, the endpoints are simply updated in the perturbation expressions. In practice it may be useful to recalculate the parameters after a few steps, but not necessarily in every branching node. New perturbation expressions should be analyzed to show that they yield reasonably efficient parameter calculations, reusability on subdomains, and convergence in the sense discussed in Section 3.3.

It is sometimes implied that the long-term goal for (MI)NLP solvers is to reach a similar level of maturity and reliability as current MILP software [22, 49]. Actively maintained commercial MILP solvers are robust and represent today's state-of-the-art in their field. A special-purpose solver is unlikely to outperform them, unless it exploits some deep combinatorial structure in the problem. Thus, an analyst can feed

a model to the solver and trust that a strong and robust attempt is made to solve it. For MINLP problems the situation is different. A simple manipulation of a constraint may determine if optimality can be verified, or even whether a feasible solution will be found or not. It falls on the users to try different model formulations according to their experience and understanding of the solution process.

A comparison from 2004, admittedly a long time ago in software development, may serve to illustrate the state of global solver reliability. Neumaier et al. benchmarked nine solvers on 1000 problems under a prescribed protocol [89, 106]. BARON (version 7.2), which is considered a state-of-the-art global solver, accepted 950 of the problem inputs. It solved 86% of the accepted problems to global optimum and correctly claimed that the global optimum was found for 68% of the problems. BARON's claims to have found a global solution were wrong 4% of the time. Other solvers also suffer from such misjudgments, it would be interesting to know if the main cause is algorithms, implementations or difficult near-degenerate problems. The same reliability comparison could be made with updated contemporary solvers to show the current situation. New underestimation methods may well make a dent in the number of unsolved problems, and problems where the global optimum was found but not verified. A representable αBB solver should be included when an implementation is made.

The question is whether global MINLP solvers can ever reach the same level of robustness as their MILP counterparts. Consider the problem [94] of maximizing $x_1 \cdot x_2^2 \cdot \ldots \cdot x_n^n$ over the simplex $x_1 + x_2 + \ldots + x_n = n$, $\boldsymbol{x} \geq \mathbf{0}$. Global solver software, such as BARON 11.5.2 or Couenne 0.4, succeed in verifying the global solution only for small n. On the other hand, a short argument with Lagrange multipliers shows that the optimal solution components have ratios $1 : 2 : \ldots : n$, which quickly yields an analytical solution for any n. This trick could be programmed and automated to catch a few special problems only to have the solver foiled by some other type of simpler-than-it-looks problem. The ultimate optimization software would try many kinds of transformations and techniques on a problem, but the possibilities combine and form an exponential number of "special" cases. There may always be a need for a trained human analyst to provide ideas and guide the process.

Bibliography

[1] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005. (18)

[2] C. S. Adjiman, I. P. Androulakis, and C. A. Floudas. A global optimization method, αBB, for general twice-differentiable constrained NLPs – II. Implementation and computational results. *Computers & Chemical Engineering*, 22(9):1159–1179, 1998. (25, 27)

[3] C. S. Adjiman, S. Dallwig, C. A. Floudas, and A. Neumaier. A global optimization method, αBB, for general twice-differentiable constrained NLPs – I. Theoretical advances. *Computers & Chemical Engineering*, 22(9):1137–1158, 1998. (24, 25, 27, 35, 37)

[4] C. S. Adjiman, I. P. Androulakis, and C. A. Floudas. Global optimization of mixed-integer nonlinear problems. *AIChE Journal*, 46(9):1769–1797, 2000. (18)

[5] A. Ahmadi, A. Olshevsky, P. Parrilo, and J. Tsitsiklis. NP-hardness of deciding convexity of quartic polynomials and related problems. *Mathematical Programming*, 137(1-2):453–476, 2013. (11, 15)

[6] I. G. Akrotirianakis and C. A. Floudas. A new class of improved convex underestimators for twice continuously differentiable constrained nlps. *Journal of Global Optimization*, 30: 367–390, December 2004. (38)

[7] I. G. Akrotirianakis and C. A. Floudas. Computational experience with a new class of convex underestimators: Box-constrained NLP problems. *Journal of Global Optimization*, 29:249–264, 2004. 10.1023/B:JOGO.0000044768.75992.10. (38)

[8] I. G. Akrotirianakis, C. A. Meyer, and C. A. Floudas. The role of the off-diagonal elements of the hessian matrix in the construction of tight convex underestimators for nonconvex functions. In *Foundations of Computer-Aided Design (FOCAPD'04)*, 2004. (32, 36)

[9] F. A. Al-Khayyal and J. E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, 8(2):273–286, 1983. (23)

[10] I. Androulakis, C. Maranas, and C. Floudas. αBB: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4):337–363, 1995. (27)

[11] K. Appel and W. Haken. Every planar map is four colorable. Part I: Discharging. *Illinois Journal of Mathematics*, 21(3):429–490, 1977. (14)

[12] K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003. (18)

[13] S. Arora. Nearly linear time approximation schemes for euclidean tsp and other geometric problems. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pp. 554–563, 1997. (10)

[14] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, May 1998. (10)

[15] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamala, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999. (5, 9)

[16] M. Baes, A. Del Pia, Y. Nesterov, S. Onn, and R. Weismantel. Minimizing lipschitz-continuous strongly convex functions over integer points in polytopes. *Mathematical Programming*, 134:305–322, 2012. (11)

[17] Barton Lab, MIT. DAEPACK website. `http://yoric.mit.edu/DAEPACK`. (27)

[18] M. Bellare and P. Rogaway. The complexity of approximating a nonlinear program. *Mathematical Programming*, 69(1–3):429–441, 1995. (11)

[19] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957. (18)

[20] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tighteningtechniques for non-convex MINLP. *Optimization Methods and Software*, 24(4-5):597–634, 2009. (18, 27)

[21] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization*. SIAM, 2001. (11)

[22] T. Berthold and A. M. Gleixner. Undercover branching. In V. Bonifaci, C. Demetrescu, and A. Marchetti-Spaccamela, editors, *Experimental Algorithms*, Volume 7933 of *Lecture Notes in Computer Science*, pp. 212–223. Springer Berlin Heidelberg, 2013. (38)

[23] A. Bompadre and A. Mitsos. Convergence rate of McCormick relaxations. *Journal of Global Optimization*, 52(1):1–28, 2012. (19, 29, 38)

[24] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. (15)

[25] S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi. A tutorial on geometric programming. *Optimization and Engineering*, 8:67–127, 2007. (11)

[26] R. S. Boyer. A mechanically proof-checked encyclopedia of mathematics: Should we build one? Can we? In A. Bundy, editor, *Automated Deduction – CADE-12*, Volume 814 of *Lecture Notes in Computer Science*, pp. 237–251. Springer Berlin Heidelberg, 1994. (14)

[27] A. Brauer. Limits for the characteristic roots of a matrix II. *Duke Mathematical Journal*, 14: 21–26, 1947. (34)

[28] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976. (9)

[29] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pp. 151–158, New York, NY, USA, 1971. (8)

[30] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301, 1965. (6)

[31] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990. (6)

[32] G. Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical Programming*, 112(1):3–44, 2008. (16)

[33] CVX Research, Inc. CVX: Matlab software for disciplined convex programming, version 2.0 beta. `http://cvxr.com/cvx`, September 2012. (15, 33)

[34] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 1965. (15)

[35] P. D'Alberto and A. Nicolau. Using recursion to boost ATLAS's performance. In J. Labarta, K. Joe, and T. Sato, editors, *High-Performance Computing*, Volume 4759 of *Lecture Notes in Computer Science*, pp. 142–151. Springer Berlin Heidelberg, 2008. (6)

[36] A. Drud. Conopt solver manual. URL `http://www.gams.com/dd/docs/solvers/conopt.pdf`. (32)

[37] K. Du and R. Kearfott. The cluster problem in multivariate global optimization. *Journal of Global Optimization*, 5(3):253–265, 1994. (18)

[38] M. A. Duran and I. E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36(3):307–339, 1986. (15)

[39] J. E. Falk and R. M. Soland. An algorithm for separable nonconvex programming problems. *Management Science*, 15(9):550–569, 1969. (15)

[40] C. A. Floudas. *Deterministic Global Optimization*. Kluwer Academic Publishers, 2000. (12, 14, 16, 25, 27, 33)

[41] A. S. Fraenkel and D. Lichtenstein. Computing a perfect strategy for $n \times n$ chess requires time exponential in n. *Journal of Combinatorial Theory, Series A*, 31(2):199–214, 1981. (7)

[42] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979. (5, 7, 8, 12)

[43] E. P. Gatzke, J. E. Tolsma, and P. I. Barton. Construction of convex relaxations using automated code generation techniques. *Optimization and Engineering*, 3:305–326, 2002. (27)

[44] A. M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972. (15)

[45] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6): 1115–1145, 1995. (12, 21)

[46] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008. (10)

[47] G. Gonthier. Formal proof – the four-color theorem. *Notices of the American Mathematical Society*, 55(11):1382–1393, 2008. (14)

[48] H. J. Greenberg. *Myths and Counterexamples in Mathematical Programming*. INFORMS Computing Society, http://glossary.computing.society.informs.org, February 2010. (ongoing, first posted October 2008). (13)

[49] I. E. Grossmann, J. Viswanathan, A. Vecchietti, R. Raman, and E. Kalvelagen. DICOPT manual. URL http://www.gams.com/dd/docs/solvers/dicopt.pdf. (38)

[50] Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2013. URL http://www.gurobi.com. (32)

[51] T. Hales. Formal proof. *Notices of the American Mathematical Society*, 55(11):1370–1380, 2008. (14)

[52] T. C. Hales. A proof of the Kepler conjecture. *Annals of Mathematics*, 162(3):1065–1185, 2005. (14)

[53] E. Hansen and G. W. Walster. *Global Optimization using Interval Analysis, Second Edition*. Marcel Dekker, Inc., 2004. (16)

[54] M. Heideman, D. Johnson, and C. Burrus. Gauss and the history of the fast fourier transform. *ASSP Magazine, IEEE*, 1(4):14–21, 1984. (6)

[55] L. A. Hemaspaandra. Complexity theory column 36. *SIGACT News*, 33(2):34–47, June 2002. (8)

[56] M. Hladík. On the efficient Gerschgorin inclusion usage in the global optimization αBB method. *Journal of Global Optimization*, pp. 1–19, 2014. DOI: 10.1007/s10898-014-0161-7. (35, 36, 37)

[57] M. Hladík, D. Daney, and E. P. Tsigaridas. Bounds on eigenvalues and singular values of interval matrices. Rapport de recherche 1234, Centre de recherche INRIA, October 2008. (35)

[58] M. Hladík, D. Daney, and E. P. Tsigaridas. An algorithm for addressing the real interval eigenvalue problem. *Journal of Computational and Applied Mathematics*, 235(8):2715–2730, 2011. (35)

[59] J. Hooker. Logic, optimization, and constraint programming. *INFORMS Journal on Computing*, 14(4):295–321, 2002. (18)

[60] R. Horst and H. Tuy. *Global Optimization: Deterministic Approaches*. Springer, 1996. (16)

[61] J. Hromkovič. *Algorithmics for Hard Problems*. Springer, 2001. (11, 12)

[62] R. Jeroslow. Trivial integer programs unsolvable by branch-and-bound. *Mathematical Programming*, 6(1):105–109, 1974. (18)

[63] R. M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pp. 85–103. Plenum Press, 1972. (8)

[64] R. Kaye. Minesweeper is NP-complete. *The Mathematical Intelligencer*, 22(2):9–15, 2000. (8)

[65] B. Kearfott and K. Du. The cluster problem in global optimization: the univariate case. In R. Albrecht, G. Alefeld, and H. Stetter, editors, *Validation Numerics*, Volume 9 of *Computing Supplementum*, pp. 117–127. Springer Vienna, 1993. (18)

[66] A. Khajavirad and N. V. Sahinidis. Convex envelopes generated from finitely many compact convex sets. *Mathematical Programming*, 137(1–2):371–408, 2013. (22)

[67] M. Köppe. On the complexity of nonlinear mixed-integer optimization. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, *The IMA Volumes in Mathematics and its Applications*, Volume 154. Springer, 2011. (11)

[68] R. E. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1): 155–171, January 1975. (8)

[69] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960. (15)

[70] J. Lasserre and T. Thanh. Convex underestimators of polynomials. *Journal of Global Optimization*, 56(1):1–25, 2013. (33, 36)

[71] L. Liberti. *Reformulation and Convex Relaxation Techniques for Global Optimization*. PhD thesis, Imperial College London, 2004. (21)

[72] R. Lougee-Heimer. The Common Optimization INterface for Operations Research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development*, 47(1):57–66, 2003. (27, 38)

[73] A. Lundell. *Transformation Techniques for Signomial Functions in Global Optimization*. PhD thesis, Åbo Akademi University, 2009. (31)

[74] A. Lundell, J. Westerlund, and T. Westerlund. Some transformation techniques with applications in global optimization. *Journal of Global Optimization*, 43(2):391–402, 2009. (31)

[75] A. Lundell, A. Skjäl, and T. Westerlund. A reformulation framework for global optimization. *Journal of Global Optimization*, 57(1):115–141, 2013. (3, 32)

[76] R. Martí. Multi-start methods. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, Volume 57 of *International Series in Operations Research & Management Science*, pp. 355–368. Springer US, 2003. (13)

[77] G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I – Convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976. (23, 25, 26)

[78] C. A. Meyer and C. A. Floudas. Convex underestimation of twice continuously differentiable functions by piecewise quadratic perturbation: Spline αBB underestimators. *Journal of Global Optimization*, 32:221–258, 2005. (31)

[79] C. A. Meyer and C. A. Floudas. Convex envelopes for edge-concave functions. *Mathematical Programming*, 103:207–224, 2005. (23)

[80] R. Misener and C. A. Floudas. A framework for globally optimizing mixed-integer signomial programs. *Journal of Optimization Theory and Applications*, 2013. DOI: 10.1007/s10957-013-0396-3. (27)

[81] R. Misener and C. Floudas. GloMIQO: Global mixed-integer quadratic optimizer. *Journal of Global Optimization*, 57(1):3–50, 2013. (27)

[82] J. E. Mitchell. Branch and cut. In J. J. Cochran, L. A. Cox, P. Keskinocak, J. P. Kharoufeh, and J. C. Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., 2011. (18)

[83] A. Mitsos, B. Chachuat, and P. I. Barton. McCormick-based relaxations of algorithms. *SIAM Journal on Optimization*, 20(2):573–601, 2009. (25)

[84] K. Murty and S. Kabadi. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39:117–129, 1987. (11)

[85] A. Nemirovski. Lecture notes: Interior point polynomial time methods in convex programming, 2004. URL `http://www2.isye.gatech.edu/~nemirovs/Lect_IPM.pdf`. (11)

[86] Y. Nesterov and A. Nemirovski. *Interior-Point Polynomial Methods in Convex Programming*. SIAM, 1994. (11)

[87] A. Neumaier. *Interval Methods for Systems of Equations*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge, 1990. (26, 34)

[88] A. Neumaier. Complete search in continuous global optimization and constraint satisfaction. *Acta Numerica*, 13:271–369, 5 2004. (14)

[89] A. Neumaier, O. Shcherbina, W. Huyer, and T. Vinkó. A comparison of complete global optimization solvers. *Mathematical Programming*, 103(2):335–356, 2005. (27, 39)

[90] C. Papadimitriou and S. Vempala. On the approximability of the traveling salesman problem. *Combinatorica*, 26(1):101–120, 2006. (9, 10)

[91] P. M. Pardalos and G. Schnitger. Checking local optimality in constrained quadratic programming is NP-hard. *Operations Research Letters*, 7(1):33–35, 1988. (11)

[92] P. M. Pardalos and S. A. Vavasis. Quadratic programming with one negative eigenvalue is NP-hard. *Journal of Global Optimization*, 1:15–22, 1991. (11)

[93] P. A. Parrilo. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. PhD thesis, California Institute of Technology, 2000. (33)

[94] Project Euler. Problem 190. `http://projecteuler.net`. (39)

[95] M. Putinar. Positive polynomials on compact semi-algebraic sets. *Indiana University Mathematics Journal*, 42(3):969–984, 1993. (33)

[96] I. Quesada and I. E. Grossmann. A global optimization algorithm for linear fractional and bilinear programs. *Journal of Global Optimization*, 6(1):39–76, 1995. (16)

[97] S. Robinson. Toward an optimal algorithm for matrix multiplication. *SIAM News*, 38(9), 2005. (6)

[98] R. T. Rockafellar. Lagrange multipliers and optimality. *SIAM Review*, 35:183–283, 1993. (11)

[99] J. Rohn. Bounds on eigenvalues of interval matrices. *Zeitschrift für Angewandte Mathematik und Mechanik*, 78:1049–1050, 1998. (34)

[100] J. Rohn. A handbook of results on interval linear problems. Technical Report V-1163, Institute of Computer Science, Academy of Sciences of the Czech Republic, `http://uivtx.cs.cas.cz/~rohn/publist/!aahandbook.pdf`, 2012. (11, 34)

[101] G. Rote. The convergence rate of the sandwich algorithm for approximating convex functions. *Computing*, 48(3–4):337–361, 1992. (27)

[102] H. S. Ryoo and N. V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–138, 1996. (18)

[103] H. Ryoo and N. Sahinidis. Global optimization of nonconvex NLPs and MINLPs with applications in process design. *Computers & Chemical Engineering*, 19(5):551–566, 1995. (16, 25)

[104] S. Sahni. Computationally related problems. *SIAM Journal on Computing*, 3:262–279, 1974. (11)

[105] A. Scott, U. Stege, and I. Rooij. Minesweeper may not be NP-complete but is hard nonetheless. *The Mathematical Intelligencer*, 33(4):5–17, 2011. (8)

[106] O. Shcherbina, A. Neumaier, D. Sam-Haroud, X.-H. Vu, and T.-V. Nguyen. Benchmarking global optimization and constraint satisfaction codes. In C. Bliek, C. Jermann, and A. Neumaier, editors, *Global Optimization and Constraint Satisfaction*, Volume 2861 of *Lecture Notes in Computer Science*, pp. 211–222. Springer Berlin Heidelberg, 2003. (39)

[107] H. D. Sherali and W. P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Springer, 1998. (21)

[108] E. Smith and C. Pantelides. Global optimization of general process models. In I. E. Grossmann, editor, *Global Optimization in Engineering Design*, pp. 355–386. Kluwer Academic Publishers, 1996. (25)

[109] E. Smith and C. Pantelides. A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs. *Computers & Chemical Engineering*, 23: 457–478, 1999. (16)

[110] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969. (6)

[111] G. Szpiro. Mathematics: Does the proof stack up? *Nature*, 424(6944):12–13, July 2003. (14)

[112] F. Tardella. On the existence of polyhedral convex envelopes. In C. A. Floudas and P. M. Pardalos, editors, *Frontiers in Global Optimization*. Kluwer Academic Publishers, 2003. (21, 23)

[113] M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming – Theory, Algorithms, Software, and applications*. Kluwer Academic Publishers, 2002. (18, 22, 23, 27)

[114] M. Tawarmalani and N. V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99:563–591, 2004. (18)

[115] B. Trakhtenbrot. A survey of russian approaches to perebor (brute-force searches) algorithms. *Annals of the History of Computing*, 6(4):384–400, 1984. (8)

[116] A. Tsoukalas and A. Mitsos. Multi-variate McCormick relaxations. (unpublished), 2013. URL `http://www.optimization-online.org/DB_FILE/2012/05/3473.pdf`. (25, 26)

[117] R. S. Varga. *Geršgorin and His Circles*. Springer, 2004. (35)

[118] S. A. Vavasis. Quadratic programming is in NP. *Information Processing Letters*, 36:73–77, 1990. (11)

[119] S. A. Vavasis. Approximation algorithms for indefinite quadratic programming. *Mathematical Programming*, 57(1-3):279–311, 1992. (9, 11)

[120] S. A. Vavasis. Complexity issues in global optimization: A survey. In *Handbook of Global Optimization*, pp. 27–41. Kluwer, 1995. (11)

[121] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001. (5)

[122] A. Wechsung, S. Schaber, and P. Barton. The cluster problem revisited. *Journal of Global Optimization*, 58(3):429–438, 2014. (19)

[123] I. Wegener. On the expected runtime and the success probability of evolutionary algorithms. In U. Brandes and D. Wagner, editors, *Graph-Theoretic Concepts in Computer Science*, Volume 1928 of *Lecture Notes in Computer Science*, pp. 1–10. Springer Berlin Heidelberg, 2000. (12)

[124] I. Wegener. *Complexity Theory – Exploring the Limits of Efficient Algorithms*. Springer-Verlag, 2005. (5, 7, 8, 10)

[125] T. Westerlund. Some transformation techniques in global optimization. In L. Liberti and N. Maculan, editors, *Global Optimization: From Theory to Implementation*, Volume 84 of *Nonconvex Optimization and its Applications*, pp. 47–74. Springer, 2005. (31)

[126] T. Westerlund and F. Pettersson. An extended cutting plane method for solving convex MINLP problems. *Computers & Chemical Engineering*, 19:131–136, 1995. (15)

[127] T. Westerlund, H. Skrifvars, I. Harjunkoski, and R. Pörn. An extended cutting plane method for a class of non-convex MINLP problems. *Computers & Chemical Engineering*, 22(3):357–365, 1998. (15)

[128] T. Westerlund, A. Lundell, and J. Westerlund. Some notes on convex relaxations. *AIDIC Conference Series*, 10:383–392, 2011. (21)

[129] F. Wiedijk. The QED manifesto revisited. *Studies in Logic, Grammar and Rhetoric*, 10(23): 121–133, 2007. (14)

[130] A. Zanette, M. Fischetti, and E. Balas. Can pure cutting plane algorithms work? In A. Lodi, A. Panconesi, and G. Rinaldi, editors, *Integer Programming and Combinatorial Optimization*, Volume 5035 of *Lecture Notes in Computer Science*, pp. 416–434. Springer Berlin Heidelberg, 2008. (16)

[131] S. Zlobec. On the Liu–Floudas convexification of smooth programs. *Journal of Global Optimization*, 32:401–407, 2005. (24)

[132] S. Zlobec. Characterization of convexifiable functions. *Optimization*, 55(3):251–261, 2006. (24)