2014-01-01

Contributions To Global Optimization Using Interval Methods And Speculation

Angel Fernando Garcia Contreras
University of Texas at El Paso, afgarciacontreras@miners.utep.edu

CONTRIBUTIONS TO GLOBAL OPTIMIZATION USING INTERVAL METHODS AND SPECULATION

ANGEL FERNANDO GARCIA CONTRERAS

Department of Computer Science

APPROVED:

Martine Ceberio, Chair, Ph.D.

Vladik Kreinovich, Ph.D.

Heidi Taboada-Jimenez, Ph.D.

Charles Ambler, Ph.D.
Dean of the Graduate School

CONTRIBUTIONS TO GLOBAL OPTIMIZATION USING INTERVAL METHODS AND SPECULATION

by

ANGEL FERNANDO GARCIA CONTRERAS

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Computer Science

THE UNIVERSITY OF TEXAS AT EL PASO

December 2014

to my

FAMILY, FRIENDS and COLLEAGUES

Acknowledgements

I would like to thank my advisor, Dr. Martine Ceberio, for giving me the opportunity to do research in Computer Science at The University of Texas at El Paso. She has been patient and encouraging, always finding ways in which we can become more engaged in our own research, all with the ultimate purpose of becoming experts that can make meaningful contributions in our fields. It has been a long road for both of us, full of complications that I never imagined I would find. Her guidance, support, and faith in my work is one of the things that has kept me going even in the face of stressful time and resources constraints, showing me a career path I did not think was mine.

I also would like to thank all my fellow team mates at the Constraint Research and Reading Group at The University of Texas at El Paso, whose support, dedication and inspiration have proven to be invaluable in the completion of this work. In particular, I would like to thank Luis Gutierrez, whose assitance in brainstorming, troubleshooting code, running tests, and reviewing this work brought some peace of mind and focus during the most difficult times.

Next, I would like to thank my family: My father, Angel Garcia Barrientos, my mother, Patricia Contreras, and my two sisters, Sarah and Alexandra. Their unwavering support helped me through the hardest times, motivating me to continue my career as a computer scientist and pursue higher education. I'm proud to have you as family, and this accomplishment would be impossible without your love and support.

Finally, I would like to thank the University of Texas at El Paso, for all the opportunities they give to students to promote and advance science and technology in El Paso and the world.

NOTE: This thesis was submitted to my Supervising Committee on the September 2, 2014.

Abstract

Most electronic devices we are familiar with, such as cell phones and computers, are small and require similarly small electronic components arranged and connected in small areas. Finding the right size and arrangement of the components inside a device can be a challenge. The manufacturing process of the components limits their possible size, some components have specific needs to operate at a certain speed, and the total area of the device is also limited. In portable devices, these designs have one important objective: that the entire device consumes the minimum amount of electricity possible, so the device can keep functioning for a longer time without recharging its battery. Engineers could try multiple designs and see which one is best. This unstructured approach would be inefficient, require a lot of time, and still likely not guarantee that the best configuration has been identified.

Instead, we can express the design restrictions as a series of mathematical equations, the parameters of the design as a set of variables, and the energy as an objective function to be minimized. This is an optimization problem, a category of problems represented as mathematical models in which we seek the minimum (or maximum) value of an objective function, while possibly meeting some constraints / requirements.

In order to solve an optimization problem, search algorithms are needed. Local search focuses on making slight adjustments to the values of the parameters to get progressively better objective function values, and stopping the search when the objective value cannot be improved by any nearby parameter values. Local search algorithms rely on an initial guess of the parameter values to converge to the closest minimum of the objective function. Because of this, they cannot guarantee that the solution is the overall minimum (a global optimum). Global search uses techniques that expand the range of possible values the parameters to overcome the drawback of local search. Global search techniques trade off execution time for improved accuracy, taking a longer time to find the global optimum.

In this thesis, a global optimization search algorithm is introduced. The main goal

of this algorithm is to guarantee an objective function value that is a global optimum. Interval are used to model the search space and guarantee an exhaustive search, hence a global result.

Two search algorithms were developed. The first algorithm uses speculation over the interval range of the objective function, by placing "bets" on the expected minimum value until finding it or proving it is not correct. The second algorithm is an enhancement of the first one, improving the convergence of the upper bound of the objective function by using derivative-based local search techniques for constrained and unconstrained optimization.

Our results show promise. The first algorithm finds good interval enclosures of the objective function for both constrained and unconstrained problems. The second algorithm shows improvements on unconstrained optimization by finding a better upper bound of the objective functions evaluation interval in less time than the first algorithm. These contributions will help in developing more efficient and guaranteed global optimizers for complex optimization problems.

Table of Contents

Page

Acknowledgements iv

Abstract v

Table of Contents vii

Chapter

1 Introduction 1

1.1 Decision-making 1

1.2 Contributions 3

1.3 Outline of this thesis 4

2 Preliminary notions and related work 5

2.1 Definitions 5

2.1.1 Constraint satisfaction problem 5

2.1.2 Optimization problems 7

2.2 Intervals 8

2.2.1 Examples of the need for robust computation 9

2.2.2 Challenges with real number computations 10

2.2.3 Real intervals and interval arithmetic 10

2.3 Constraint satisfaction solving techniques 14

2.3.1 Consistency techniques 15

2.3.2 Solving constraint satisfaction problems 17

2.4 Optimization techniques and algorithms 18

2.4.1 Local optimization and global optimization 18

2.4.2 Local search algorithms 19

2.4.3 Global search algorithms 21

2.5 Related work: existing interval analysis, constraint solving and optimization libraries . . . 26
2.5.1 ALIAS-C++ . . . 26
2.5.2 IBEX . . . 26
2.5.3 BARON . . . 27
2.5.4 MINOS . . . 27
2.5.5 DONLP2 . . . 27
2.5.6 RealPaver . . . 28
3 The speculative algorithm: original implementation and results . . . 29
3.1 Speculative algorithm . . . 29
3.2 Methodology . . . 31
3.3 Results . . . 32
3.3.1 Analysis of results . . . 34
3.3.2 Reflection on our proposed approach . . . 34
4 Improvements on the speculative algorithm . . . 40
4.1 Unconstrained optimization . . . 40
4.2 Constrained optimization . . . 41
4.2.1 Karush-Kuhn-Tucker conditions . . . 43
4.3 Results . . . 45
4.3.1 Analysis of results . . . 46
4.3.2 Reflection on our proposed approach . . . 46
5 Concluding remarks . . . 52
5.1 Conclusions . . . 52
5.2 Future work . . . 53
Bibliography . . . 55
Curriculum vita . . . 61

Chapter 1

Introduction

1.1 Decision-making

Our daily lives require making decisions. As a simple example, we daily select the clothes we are going to wear. In order to make the right decision, we need to examine how appropriate the clothes will be for the anticipated situation: wearing a bathing suit is appropriate for going to the pool or the beach, but certainly not for going to a funeral or for skiing. Criteria such as formality, weather, and safety are the most basic ones; others might take into account personal preferences. These simple decisions are taken through logic, and the process is carried out entirely inside the decision-maker's mind.

Some decisions are not that easy to make without aid; sometimes because there are too many parameters to decide on, sometimes because these parameters are inter-related in ways that make decisions hard. For example, a student considering to drop out of college is influenced by poor academic performance, lack of preparation, or economic troubles, to name a few. Each one of these factors can be influenced by others, such as prior academic achievements, economic situation, employment situation, or even whether the student is a first-generation college student [7, 59].

Decision-making also takes place in science and industry, for example the design of electronic devices. They have become important in our daily lives, we use computers and cell phones for convenience and communication and we find electronics even in places we cannot see. They are all around us yet we ignore many of the aspects behind the manufacturing process of these devices, such as the design and arrangement of the electronic components inside the device. The manufacturing process of these small components limits

their possible size and how much energy they consume. Different components need to work within certain operation speeds, or they will not function reliably. And then there is the size of the device itself, which limits the number of components and the distance between them [6].

Based on the purpose of the device, the engineer in charge of designing it must take all these requirements into consideration to generate a suitable candidate to manufacture. One common challenge lies in the power consumption of the device, as companies try to provide portable electronics that can last longer without recharging. In that case, the engineer must also find the component parameters that result in the least power consumptio n[6]. Considering that there is an infinite number of possible sizes, speeds and arrangements of components, the engineer cannot test them all. He wants a design that is the most energy efficient of all designs.

As a result of the complexity of these problems, it either takes long to make decisions properly or we settle for approximate solutions as "good" decisions. Decisions where we have to find the value of parameters that satisfy properties (or constraints) are called: *constraint solving problems* or parameter estimation problems [1]. We can find these kinds of problems in many areas. For example, the system that schedules the times for astronomical observations in the Hubble Space Telescope uses constraint satisfaction techniques to determine when an object can be observed depending on the position of the Earth, the sun, and the telescope itself. Every year, the telescope carries out tens of thousands of exposures for thousands of astronomical objects. Scheduling must be done months in advance, but there are constraints that involve timeframes that range from minutes to years. It is a perfect example of a complex constraint satisfaction problem that can only be solved using computations [48, 34].

Sometimes, there exist multiple solutions to a decision/parameter estimation problem and we might be interested in: finding only one solution, finding all solutions, finding the best solution. When looking for all solutions or the guaranteed best solution, we say that the decision problems are global problems. For example, let us consider the work of an

investment firm. An agent has a portfolio of multiple assets, and each asset can receive a certain investment. However, the total amount that can be invested among all assets is limited, and there is a minimum expected return for the entire portfolio. Investing is a risky business, so one approach the agent can take is to try to find the combination of investments that minimizes the risk in his portfolio, while maintaining the minimum returns and budget constraints of the portfolio [6, 49, 17].

These *optimization* methods have a brief history. The first methods that solved linear optimization problems were developed in the 1940s. Since then, with the advent of faster computing, mathematical optimization has found applications in multiple fields, such as engineering, electronics, industrial design and economics, to name a few [54].

The methods used to solve optimization problems are varied, involving techniques from disciplines such as decision analysis, system analysis, control theory, game theory, constraint programming, artificial intelligence, decision making, and so on [20].

The complexity of modern optimization problems can be daunting. For example, if we want to build a predictive model that can measure the quality of a software program based on its attributes, we can use existing data of the attributes of existing software program and how these programs were evaluated by experts. The mathematical model has tens to hundreds of variables and constraints, and a polynomial objective function with hundreds of mathematical terms, all in an attempt to find the parameters that define a model that minimizes the discrepancies that may exist between the expert evaluations [9].

In the work that we present in this thesis, we focus on the global best solution of problems, or *global optimization problems*, and their algorithmic solutions.

1.2 Contributions

In this thesis, we address challenges posed by global optimization of potentially nonlinear functions, over continuous domains, with or without additional constraints.

The main challenge is the size of the search space of interest. In this work, regardless

of the search space size, we focus the search on one dimension only: the range of the objective function. The driver of our search, in a way, looks for a value of the objective function rather than looking for values of the parameters. We call this part of our approach speculative because we always make "bets" on the expected minimum value and carry on with it until we find it or we conclude it was not correct.

Another approach is to enhance the convergence of our search algorithms onto the solutions. We achieve this through good contractors (operators that shrink the search space, deleting areas known not to contain solutions). We add extra constraints (hence contractors) to the original problem in the form of a known low value of the objective function obtained via a local algorithm. This, combined with speculation, constitutes our second optimization algorithm.

1.3 Outline of this thesis

In Chapter 2, we review preliminary notions and background necessary to understand our work: we introduce basic concepts in constraint satisfaction and optimization, present basic approaches to solve these problems, and finally review existing state-of-the-art solvers as well as current research in the area.

In Chapter 3, we present our speculative optimization algorithm, explaining its process as well as reporting preliminary results. In Chapter 4, we present ways to improve our original speculative optimization algorithm in order to achieve better time to solution and accuracy, as well as report the results of these enhancements.

We conclude and draw directions for future work in Chapter 5.

Chapter 2

Preliminary notions and related work

Global optimization is a research field that emerged along with the growth of computational power. The first computer-based optimization algorithms, developed before the 1970's, were strictly local. The first attempts at global optimization were developed in the 1970's and beyond, mostly focusing on stochastic techniques that do not guarantee finding a global solution, but will find one with high probability in the right circumstances. Truly global optimization algorithms have started to be developed and expanded only in the last 20 years, when computational power allows using these techniques to solve more complex problems [27].

In this chapter, we provide the general definitions of the types of problem that are the subject of this work. Then, we briefly introduce interval analysis, which serves as the basis of many of the techniques, algorithms and solvers that are outlined afterwards.

2.1 Definitions

2.1.1 Constraint satisfaction problem

A *constraint* is an expression that defines a relationship between a set of variables, in the form of a restriction to the possible values of the variables. A *constraint satisfaction problem* is a model designed to find any / all value assignments that fulfill a set of constraints [1, 47]. In particular, the type of constraint satisfaction problem that is relevant to this work is a *real-valued* constraint satisfaction problem in which the domain of the variables is in the real numbers, as seen in Definition 1 [1].

Definition 1 A *constraint satisfaction problem* $CSP = (X, D, C)$ is a problem that contains a set of n variables

$$X = \{x_1, \ldots, x_n\}$$

with respective domain values

$$D = D_1 \times \ldots \times D_n \quad \text{where} \quad x_i \in D_i \quad \text{and} \quad \forall i, D_i \subseteq \mathbb{R}$$

and a set of m constraints

$$C = \{c_1, \ldots, c_m\} \quad \text{where}$$

$$c_k = f_k(x_1, \ldots, x_n) \bowtie g_k(x_1, \ldots, x_n), \quad \forall k \in \{1, \ldots, m\}, \quad \bowtie \in \{\leq, \geq, =\}$$

For a real-valued constraint satisfaction problem, we say a *solution* is a set of variable values within the corresponding variable domains such that all the constraints are satisfied [1]. Definition 2 shows the definition of satisfaction for a singular constraint, and Definition 3 is the solution of a constraint satisfaction problem.

Definition 2 For a constraint c, variables $X = \{x_1, \ldots, x_n\}$ with respective domains $D = \{D_1, \ldots, D_n\} \quad where \quad x_i \in D_i, \quad D_i \subset \mathbb{R}, \forall i \in \{1, \ldots, n\}$, the n-tuple $d = (d_1, \ldots, d_n)$ where $d_i \in D_i, \forall i \in \{1, \ldots, n\}$, the n-tuple d *satisfies* the constraint c if the relation

$$f(d_1, \ldots, d_n) \bowtie g(d_1, \ldots, d_n), \quad \bowtie \quad \in \{\leq, \geq, =\}$$

holds true.

Definition 3 Having a $CSP = (X, D, C)$ with a set of n variables X with respective n domains D, and m constraints C, the n-tuple $d = (d_1, \ldots, d_n)$ where $d_i \in D_i$, $\forall i \in \{1, \ldots, n\}$, is a *solution* to $CSP = (X, D, C)$, if d satisfies all the constraints in C

2.1.2 Optimization problems

Constraint satisfaction problems only require finding one or more solutions that satisfy the problem restrictions. This is just one type of problem. Other problems require finding the best of all solutions (an optimum) under a set of constraints. Definition 4 is the general form for an *optimization problem.*

Definition 4 A *numerical optimization problem* is defined by the expression

$$\begin{aligned} &\text{minimize } f(X) \\ &\quad \text{s.t.} \quad g_i(X) \leq 0, \quad i \in \{1, \ldots, m\} \\ &\qquad\quad h_k(X) = 0, \quad k \in \{1, \ldots, p\} \end{aligned}$$

where X is a set of variables

$$X = \{x_1, \ldots, x_n\}$$

with respective domain values

$$D = D_1 \times \ldots \times D_n \quad \text{where} \quad x_i \in D_i \quad \text{and} \quad \forall i, D_i \subseteq \mathbb{R}$$

Optimization problems usually can have two different types of constraints: *equality* and *inequality* constraints. A constrained optimization problem can have both types of constraints, only one type of constraint, or none at all (Known as an *unconstrained optimization* problem) [1, 6].

All optimization problems can be expressed as *minimization* problems. Maximization problems are converted into minimization problems by multiplying their objective function by -1. Figure 2.1 shows an example in which the maximum of function $f(x)$ is the minimum of $-f(x)$.

A *solution* of an optimization problem is an instantiation of the variables for which there is no better evaluation of the objective function, as seen in Definition 5 [1].

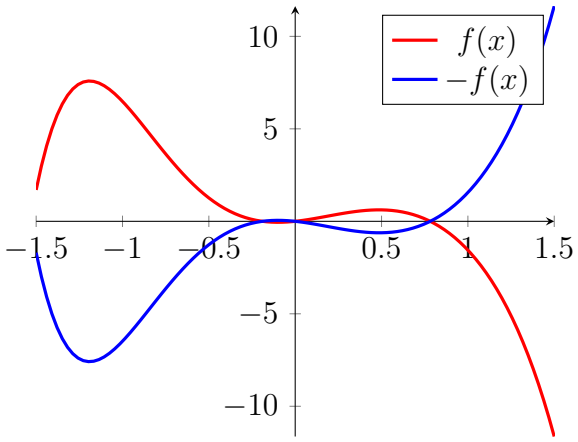

Figure 2.1: Function and negated function

Definition 5 Given the following optimization problem:

$$\begin{aligned} &\text{minimize } f(X) \\ &\quad \text{s.t.} \quad g_i(X) \leq 0, \quad i \in \{1, \ldots, m\} \\ &\qquad\quad h_k(X) = 0, \quad k \in \{1, \ldots, p\} \end{aligned}$$

where X is a set of variables $X = \{x_1, \ldots, x_n\}$ with respective domain values $D = D_1 \times \ldots \times D_n$ where $x_i \in D_i$ and $\forall i, D_i \subseteq \mathbb{R}$, a *solution* to this problem is a set of instantiations X^* of the variables X: $X^* = (x_1^*, \ldots, x_n^*)$ where $\forall i \in \{1, \ldots, n\}$, $x_i^* \in D_i$, such that:

$$\begin{cases} f(X^*) \leq f(X), \ \forall X \in D_1 \times \cdots \times D_n \\ g_i(X^*) \leq 0, \ \forall i \in \{1, \cdots, m\} \\ h_k(X^*) = 0, \ \forall k \in \{1, \cdots, p\} \end{cases}$$

2.2 Intervals

In this section, we outline the advantages provided by using intervals and we describe the interval techniques used in constraint programming and optimization algorithms.

2.2.1 Examples of the need for robust computation

February 25, 1991. In Dharan, Saudi Arabia, in the middle of the Gulf War, **a missile battery failed to identify an Iraqi Scud missile that ended up striking an American Army barrack**, killing 28 soldiers and hurting at least 100 others. What caused the battery to fail, when it had otherwise managed to detect and shoot down all previous enemy missiles? A simple calculation error. The system used time measurements in tenths of a second, or 1/10s. Inside the computer, the calculation to create the number 1/10 is not completely accurate, creating an error of 9.5×10^{-8} every tenth of a second. After the system ran for 100 hours, the miscalculation had replicated itself by the number of seconds in those 100 hours, resulting in a calculation error of 0.34. When the Scud missile passed through the battery's radar, it was properly detected. The system then tried to predict the next position of the missile before doing a second radar reading to verify the presence of the threat. This predictive calculation used the miscalculated time fraction, so when the battery attempted to sense the missile in the predicted location, it found nothing in the sky and dismissed the threat [18, 53, 2, 12, 23].

In 1982, the Vancouver Stock Exchange instituted a new stock market index using three decimal places. This new index started with a base value of 1,000.000 and each trading operation updated it using floating-point operations. However, these calculations had one flaw: the floating-point numbers resulting from each computation were not rounded to three decimal places, but truncated to that same length, which translated into a huge loss of accuracy. After 22 months of approximately 3,000 transactions per day, the index had reached a dismal value of 524.811. After a team of analysts discovered the error, they recalculated the actual value using rounded-up floating-point numbers instead, which yielded a more reasonable index of 1098.892. In November of 1983, they fixed the system and updated the index to reflect its actual value [23, 41].

2.2.2 Challenges with real number computations

The above examples highlight the main problem in machine computations: the limited accuracy of the numerical representation of real numbers and their operations. Real numbers are defined as quantities in a continuous line. A line is made of an infinite amount of points, so there is an infinite amount of numbers between any two points in the line. In other words, there exists numbers that require an infinity-sized representation, and in computation we are forced to represent them using finite space. For example, every representation of the value of π is inaccurate, it is always rounded up to a certain number of digits [57, 37].

Real numbers are often represented within the limited amount of space in a machine as a set of numbers called *floating-point numbers*. They are one of the most efficient ways to represent and do calculations with real numbers under the limited memory of a computer. They are efficient, but not perfect, as their own finite nature forces real numbers to be rounded to their nearest floating-point representation. The number is modeled as an interval, with the real number found inside an interval with floating-point bounds. This is one of the core ideas behind *interval analysis*, a research area that develops and applies interval-based techniques that consider imprecision and uncertainty to improve the precision of machine calculations [57].

2.2.3 Real intervals and interval arithmetic

Let us start with some definitions.

Definition 6 A real *interval* $[a, b]$ is defined as follows:

$$x = [a, b] = \{r \in \mathbb{R} \mid a \leq r \leq b,\ a,\ b \in \mathbb{R}\}$$

where a is the *lower bound* of x, and b is the *upper bound* of interval x. The set $\mathbb{IR}$ contains all real intervals x [57, 18, 36, 37].

Definition 7 *The smallest possible real interval contains only one element, or* $[a, a]$*, with* $a \in \mathbb{R}$*: it is called a* canonical interval *and is used to convert individual real numbers to their interval representation [35].*

Definition 8 *Given a subset* $\rho \subseteq \mathbb{R}$*, the real interval that contains all elements of* ρ *is called the* convex hull *of* ρ*, denoted* $\square\rho$*, and is defined as follows: [36, 37]:*

$$\mathit{Hull}(\rho) = \square\rho = [\min\ \rho, \max\ \rho].$$

Definition 9 *Given a vector of intervals* $(x_1, \ldots, x_n) \in \mathbb{IR}^n$*, We denote by* X *and call* box X *the cartesian product of all the intervals in the vector [36, 37]:*

$$X = x_1 \times \ldots \times x_n.$$

Definition 10 *Given a real interval* $x = [a, b]$*, its* width *is defined as follows [36, 37]:*

$$w(x) = |a - b|.$$

Definition 11 *Given a box* $X = (x_1, \ldots, x_n)$*, its* width *is defined as follows [36, 37]:*

$$w(X) = \max_{1 \leq i \leq n} w(x_i)$$

Interval arithmetic

Interval arithmetic is an extension of the arithmetic we know over real numbers.

Combining two intervals with a given binary operator, say $\diamond \in \{+, -, \times, \div\}$, results in an interval that contain all real values resulting from applying $\diamond$ to real elements of the two respective intervals. In other words [36, 37], we should have:

$$\forall X, Y \in \mathbb{IR},\ \forall \diamond \in \{+, -, \times, \div\},\ X \diamond Y = \{x \diamond y \mid (x, y) \in X \times Y\}$$

However, when applying the above formula, we might obtain a result $X \diamond Y$ that is not an interval, i.e., not convex. This would happen if we were to, say, division any interval X by an interval Y that contains 0. In order to guarantee that intervals are closed under

arithmetic expressions, we need to adjust the above expression to the following one, which ensures that the result is always an interval [36, 37]:

$$X \diamond Y = \square(\{x \diamond y \mid (x, y) \in X \times Y\}).$$

Let us note that this formula is easily generalized to operators of any arity, including beyond arithmetic operators, defining interval analysis.

Consequently, here are the rules for basic interval arithmetic operations [36, 37]:

Definition 12

$$\begin{aligned}
\text{Addition:} \quad & [a,b] + [c,d] = [a+c, b+d] \\
\text{Substraction:} \quad & [a,b] - [c,d] = [a-d, b-c] \\
\text{Multiplication:} \quad & [a,b] \times [c,d] = [\min\{ac,ad,bc,bd\}, \max\{ac,ad,bc,bd\}] \\
\text{Division:} \quad & [a,b] \div [c,d] = \begin{cases} [\min\{ac,ad,bc,bd\}, \max\{ac,ad,bc,bd\}] & \\ \qquad\qquad\qquad\qquad \text{if } 0 \notin [c,d] & \\ [-\infty, +\infty] \qquad \text{if } 0 \in [c,d] & \end{cases} \\
\text{Exponentiation:} \quad & [a,b]^n = \begin{cases} [a^n, b^n] & \text{if } n \text{ is odd} \\ [\min(|a|,|b|)^n, \max(|a|,|b|)^n] & \text{if } n \text{ is even and } 0 \notin [a,b] \\ [0, \max(a^n, b^n)] & \text{if } n \text{ is even and } 0 \in [a,b] \end{cases}
\end{aligned}$$

Interval extensions

As mentioned earlier, intervals can be combined by general functions (beyond arithmetic): this is made possible by defining *interval extensions of functions* [57].

Definition 13 For the real-valued function $f : \mathbb{R} \to \mathbb{R}$, the interval function $f_{\mathbb{I}} : \mathbb{IR}^n \to \mathbb{IR}$ is the *interval extension* of f, if

$$\forall X \in \mathbb{IR}^n, \ \{f(x) \mid x \in X\} \subseteq f_{\mathbb{I}}(X).$$

The above definition is flexible enough that, given a real function, there exist many possible interval extensions of it. The most basic is called *natural extension*, and involves carrying out the operations in the function using interval arithmetic. This method is straightforward and has little computational overhead. However, this type of evaluation can lead to imprecise results that contains the actual result of the operation, along with additional noise that is not the result. For example, considering the real-valued function $f(x) = x - x$: its range, regardless of the domain of interest, is $\{0\}$.Its natural interval extension is defined by $f_{\mathbb{I}}(x_{\mathbb{I}}) = x_{\mathbb{I}} - x_{\mathbb{I}}$. On $x_{\mathbb{I}} = [0, 2]$, the evaluation of $f_{\mathbb{I}}$ is $[-2, 2]$. Other interval extensions exist that try to address this problem, known as overestimation (of the with of the function's range): among them are natural extensions of Taylor expansions of the original function. Often, the trade-off is computational time in exchange of increased accuracy [57, 18].

Intervals and computers = floating-point bounded intervals

All of the definitions, concepts, and techniques related to intervals seen so far involve intervals of real numbers, bounded by real numbers. Computers, however, do not use real numbers. The most commonly used type to represent real numbers is floating-point numbers. In what follows, we will focus on intervals of real numbers bounded by floating-point numbers.

Definition 14 *A* floating-point-bounded interval $[a, b]$ *is defined as follows:*

$$[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b,\ a,\ b\ \in \mathbb{F}\}$$

The set $\mathbb{IF}$ *contains all such floating-point-bounded intervals.*

Note: from now on, we will address intervals in computers. As a result, when we talk about intervals, we will mean floating-point-bounded intervals.

Definition 15 *Given a real number* $a \in \mathbb{R}$*, the smallest interval containing* a *is called a*

canonical interval *and is defined as follows:*

$$[a] = [a^-, a^+]$$

where: $a^+ = \min\{x \in \mathbb{F} \mid x \geq a\}$ *and* $a^- = \max\{x \in \mathbb{F} \mid x \leq a\}$. *As a result, if* $a \in \mathbb{F}$, $[a] = [a, a]$.

As seen earlier, the conversion from real to floating points can introduce rounding errors. In order to represent real intervals in a computer using *floating-point bounds*, it is important to deal with these rounding errors [57, 36]. In particular, when converting a real interval X to a floating-point-bounded interval Y, we must ensure that $X \subseteq Y$ to guarantee the inclusion property of interval arithmetic will still hold.

Definition 16 *The* floating-point-bounded interval $x_{\mathbb{IF}}$ *representing (in computer) a real interval* $x_{\mathbb{IR}} = [a, b]$ *where* $(a, b) \in \mathbb{R}$ *is the tightest enclosure of* $x_{\mathbb{IR}}$ *within floating-point bounds, and is given by:*

$$x_{\mathbb{IF}} = [a^-, b^+]$$

which is known as the outward rounding *of* $x_{\mathbb{R}}$.

The arithmetic operations and functions on floating-point-bounded intervals are defined similarly as for real intervals. The only difference is that at each step of computations, interval bounds are outward rounded to fit to the nearest outward floating points (downward for lower bound and upward for upper bound).

2.3 Constraint satisfaction solving techniques

In Section 2.1, we presented what constraint solving and optimization problems are. In this section, we present solving techniques for these problems.

Consistency is a property of constraint satisfaction problems that contain a solution. In contrast, a constraint satisfaction problem that is inconsistent cannot be solved.

Definition 17 *Given a constraint* (X, D, c) *where* $X = \{x_1, \ldots, x_n\}$ *and* $D = \{D_1, \ldots, D_n\}$*where* $x_i \in D_i \subset \mathbb{R}$, $\forall i \in \{1, \ldots, n\}$, *and* ρ_c *the set of all the elements in* $\mathbb{R}^n$ *that satisfy constraint* c, *then any element* $s \in \rho_c \cap D$ *is a solution of* (X, D, c) *and all elements of* $\rho_c \cap D$ *are* consistent *with* c.

Definition 18 *Given a* $CSP = (X, D, C)$ *where* $X = \{x_1, \ldots, x_n\}$, $D = \{D_1, \ldots, D_n\}$*where* $x_i \in D_i \subset \mathbb{R}$, $\forall i \in \{1, \ldots, n\}$, *and* $C = \{c_1, \ldots, c_m\}$. *Being* ρ_C *the set of all the elements in* $\mathbb{R}^n$ *that satisfy all the constraints* C. *Then, any element* $s \in \mathbb{R}^n$ *of* $\rho_c \cap D$ *is a solution of* CSP, *and all elements of* $\rho_c \cap D$ *are* consistent *for the* CSP.

Solving a CSP using intervals involves computing approximate solutions in which the domains enclose the real-numbered solution within a narrow margin. The two most important techniques that contribute to finding these solutions are *contraction* and *propagation*. Contraction uses a constraint to reduce the size of the domain while still remaining consistent. Propagation is achieved through consistency algorithms, which use contraction based on a single constraint and then propagate the reduced domain to enforce consistency on other constraints. The tecniques that incorporate contraction and propagation within a CSP solving framework are called *contractors*.

2.3.1 Consistency techniques

Hull consistency

Hull consistency is a type of consistency based on the hull of the interval domain that satisfies the constraints [5].

Definition 19 Given a constraint c, a box of intervals X, and ρ_c the set of all the elements in $\mathbb{R}^n$ that satisfy constraint c, then the constraint c is said to be *hull consistent* w.r.t. the box X if and only if $X = \text{Hull}(\rho_c \cap X)$.

There are multiple techniques that enforce hull consistency. We focus on a specific hull consistency technique: HC4, which enforces hull consistency by iterating through each

constraint on a CSP and applies the HC4Revise contraction method. HC4Revise uses the original constraints of the CSP, represented as evaluation trees. HC4Revise has two main steps: the forward evaluation step which evaluates the constraint using interval extensions by traversing the tree of the constraint from leaves to roots, and the backward propagation step in which the values obtained from the forward evaluation are used to contract the domain by traversing the nodes of three from root to leaves [5].

Box consistency

Box consistency focuses on ensuring that the canonical intervals in the bounds of a domain box are consistent [4, 15].

Definition 20 A constraint c is said to be *box consistent* w.r.t. to a box $X = X_1 \times \ldots \times X_n$, and a given interval extension if and only if $X_i = \text{Hull}(\{r_i \in X_i | (X_1, \ldots, X_{i-1}, \text{Hull}(\{r_i\}), X_{i+1}, \ldots, X_n)$ satisfies $c\})$, $\forall i \in \{1, \ldots, n\}$.

BC3Revise is an algorithm that enforces consistency over a n-ary constraint c. It creates a set of univariate interval constraints $C_1^B, \ldots, C_n^B$ from the interval extension C of c. Each interval constraint is associated with a contractor by replacing all the variables but one with their interval domains. Each contractor reduces the domain of the variable by computing the leftmost and rightmost canonical intervals for which C_k^B holds [5]. Algorithm BC3 incorporates BC3Revise to propagate the contracted domains over multiple domains.

BC5 is an algorithm that combines hull and box consistency to provide better contractions. HC4 processes constraints with single occurances of variables, while BC3 processes constraints with multiple occurances of variables. It also uses interval Gauss-Seidel method to solve a square system created from the original CSP using formal methods, if such system can be generated [15].

Interval Newton methods

Interval Newton methods are local techniques that find tight bounds on the solution of linear and nonlinear systems of equations where the interval domain is a good approximation to the solution. For larger domains, the algorithm can reduce the size of the domain and prove the existence of a solution [27].

Interval Newton generates a linear system of equations using interval matrices and a starting point contained within the initial box. The Newton operator maps an interval vector (the domain) with a point vector (the starting point) to approximate a solution box that is incorporated into the system of linear equations to be solved based on the selected starting point [27].

The most common operator uses Taylor expansions to iteratively create a linear problem, and solves it using any technique that handles linear problems, such as interval Gauss-Seidel method [36, 19, 39].

2.3.2 Solving constraint satisfaction problems

A *solution* to a constraint satisfaction problem using interval domains is defined as a narrow domain that is consistent with all constraints. The width of this domain depends on the desired precision, with smaller values representing increased precision at the cost of more processing time.

Consistency techniques alone cannot find a solution, they can only guarantee that they enclose all the solutions in the domain.

Constraint satisfaction problems can have more than one solution inside their domain. Even after applying consistency techniques to remove the parts of the domain that contain no solutions, we still have not found all unique solutions. We need to *search* for those solutions using interval techniques and consistency techniques [1].

We are interested in solving constraint satisfaction problems with interval domains. To search through the domain, we use an iterative technique based on *splitting* the domain. If a

problem cannot be solved using consistency techniques alone – that is, the interval domain is too wide, the search algorithm selects a variable, bisects its interval value it into two new intervals, and creates two new subdomains using the rest of the domain. The algorithm attempts to reduce one of the new subdomains using consistency techniques, and splitting the domain again if a solution of a certain precision was not found or discarding that portion of the domain if the consistency techniques determine there is no solution in that subdomain. This iterative process searches through all the domain, removing subdomains that do not contain solutions and contracting the remaining subdomains until they enclose a solution within the desired precision [1, 56].

2.4 Optimization techniques and algorithms

2.4.1 Local optimization and global optimization

Based on the type of search, we distinguish two types of optimization algorithms: Local optimization and global optimization.

Local optimization

Definition 21 Given an optimization problem

$$\begin{aligned} &\text{minimize } f(X) \\ &\text{s.t.} \quad g_i(X) \leq 0, \quad i \in \{1, \ldots, m\} \\ &\qquad h_k(X) = 0, \quad k \in \{1, \ldots, p\} \end{aligned}$$

and an initial point $x_0 \in X$, we say that local optimization is the process of finding the minimal value of f in the neighborhood of x_0, or finding the point $x^* \in X$ such that

$$\exists V \text{ in the neighborhood of } x_0 \ / \ x^* \in V \text{ and } f(y) = \min\{f(x), x \in V\}$$

Any point x^* of X that satisfies this condition is a *local minimum* of f in the neighborhood of $x_0 \in X$. Local optimization algorithms can find local minima reliably, given a

good initial point x_0. Their main drawback is their reliance on a good initial point. If this value is not available, or is not close to the minimum, the performance of the algorithm suffers, taking longer to solve the problem or not solving it at all. And this is without considering whether the problem solver needs just a good local solution, or the best global solution: Local optimization methods can find global optima if the starting point is close, but otherwise cannot guarantee that the result found is a global optimum.

Global optimization

Definition 22 Given an optimization problem

$$\begin{aligned} &\text{minimize } f(X) \\ &\text{s.t.} \quad g_i(X) \leq 0, \quad i \in \{1, \ldots, m\} \\ &\qquad h_k(X) = 0, \quad k \in \{1, \ldots, p\} \end{aligned}$$

, we say that global optimization is the process of finding the set X^* of $x^* \in X$ such that

$$\forall x^* \in X^*,\ f(x^*) = \min\{f(x), x \in X\}$$

Any element of X that satisfies this condition is a *global minima* of f on X. Global optimization algorithms do not rely on an initial point, and instead search through the *entire domain* of X. This *complete search* guarantees the globality of the results. However, searching through entire domains is time and resource-consuming.

2.4.2 Local search algorithms

Local optimization algorithms get that name because they narrow the scope of their search to a local area. They tend to be fast, yet often rely on a good starting point for their search in order to find an optimum, and they cannot guarantee that the minimum found is the global minimum.

Newton-based methods

The Newton-Rhapson method is a well-known technique used to find the domain values for which a function returns zero by using progressive approximations using the first derivative of the function. By applying this method to the first-order derivative of the function (And using the second-order derivative for the approximation), the Newton-Rhapson method can find candidate solutions to an unconstrained problem [60, 3].

Quasi-Newton methods take the basic idea of the Newton method by incorporating different methods that approximate the values of the second derivative using the objective function and the first derivative. This category of algorithms has good performance, but is still considered local search algorithms [60, 13].

Gradient-descent method

Another well-known local method is the *gradient descent*. Also known as the method of steepest descent, this derivative-based optimization algorithm calculates each new point using the first-order derivative (gradient) of the objective function. The gradient is used to approximate the rate at which the slope of the objective function is descending to get a new point closer to a minimum [60, 3].

Direct local-search methods

Direct local search methods use heuristics to generate a set of candidate solutions that are iteratively tested and improved upon. The difference between the many algorithms that make up this category is the heuristic to generate the candidates, how the candidates are discarded, and how they are improved. For example, the *mesh adaptive direct search methods* create a mesh of certain number of regions in the domain, and select sample points inside the regions defined as such. After sampling those points, the surroundings of each point are polled in search for a descending direction, which would point towards a minimum value. This method shows convergence to local minima, and can handle constraints by

either rejecting points that violate the constraints, or allow certain points in which the rate at which the constraints are violated decreases on successive iterations [46, 22].

Model-based search methods

For many optimization problems, it is possible to create an alternative model based on the original optimization problem. This *surrogate model* is similar to the original, but is much easier to solve. Surrogate models are not globally accurate, and solving them is not guaranteed to provide a solution to the original problem [46, 30, 45].

This forms the basis of *model-based search methods*. These algorithms create an initial surrogate model around a specific point, and proceed to use another optimization algorithm to solve it. The solution found is evaluated on the original problem, if it is not a solution, then the surrogate model is updated based on the discrepancy. Each iteration the surrogate model is updated, solved and compared until an optimal solution for the original is found [46, 30, 45].

The accuracy of these algorithms relies on the fidelity of the surrogate, or how closely the model resembles the original problem, and the optimization algorithm used to solve it. A high-fidelity model is harder to create and not always preferable. Global algorithms employ more computation time, which is why they are not used to solve iterative surrogate models [46, 30, 45].

2.4.3 Global search algorithms

Global search algorithms consider the entire domain in their search. Domains can be quite large, so they must rely on heuristics to improve their search and cover the entire domain.

Stochastic search algorithms

The idea behind *stochastic search algorithms* is to use a non-deterministic element in the search, which allows them to explore more areas of the search space that might not be as

optimal but lead closer to the global optimum. These algorithms search through the entire domain; however, they rely on individual points and stochastic computations, and cannot guarantee that their result is a global optimum [46].

The *genetic algorithm* is inspired by the workings of genetics and natural selection. The algorithm generates a set of candidate solutions treated as members of a population. Each member of the population is a point with variable values that will be treated as genetic code. Every iteration of the algorithm is a generation, in which these members will be used to create new members that try to improve over the previous generation's objective function values by randomly recombining and possibly mutating genetic information, removing from the group the worst candidate solutions and replacing them with the new members. The method requires defining the parameters for the genetic population model: population size, number of generations, number of rejected population members per generation, and genetic recombination and mutation strategies. In order to ensure convergence to a minimum value requires playing with these values and strategies, which may be drastically different from problem to problem [46] [11].

Related to evolutionary algorithms, *swarm algorithms* also treat subproblems as members of a population, but instead of recombining them with each other, the algorithms try to model the behavior of a group of animals. After an initial point assignment, en each iteration of the algorithm each individual of the population changes their assigned point based on their and other members' objective value information. Optimality is determined when enough members of the population are in a point that cannot be improved anymore. The difference between the algorithms in this category is how each individual member of the population handles its search in relation to the others. In particle swarm optimization, each member has a velocity and acceleration, and change these values based on the fitness of nearby particles, accelerating towards the best values in the group. In the bee colony algorithm, individual members are divided by jobs as scouts that are always exploring the global domain, and employed and onlooker bees that do local search in a more reduced area based on the scout bees' findings. The agents in ant colony optimization record their

domain and objective function values as they traverse the domain, simulating the paths of pheromones left by ants as they forage for food. The downside of these algorithms is the same as the genetic algorithm, which is finding the right combination of parameters that ensure finding an optimum without consuming too much time and resources [46, 29, 44, 32, 43].

Deterministic search: Branch-and-bound

Deterministic search comprises algorithms using global techniques that consistently produce the same results under the same problem. The most well-known type of deterministic algorithm is *branch-and-bound*. This algorithm uses a divide-and-conquer approach, splitting the original problem into subproblems or *branches* that are processed iteratively, and using each branch to find new *bounds* on the objective function value. A complete branch-and-bound algorithm searches through the entire domain to find all global optima [46, 10]. Algorithm 1 shows the general steps of the branch-and-bound technique.

Branch-and-prune

This technique is similar to branch-and-bound, as they both create a search tree of subproblems. Branch-and-prune takes advantage of the constraints in the problem to validate if the domain of the problem is consistent and remove (prune) the parts of it that are inconsistent [58], and to divide (branch) the problem if it is consistent but is not precise enough. Algorithm 2 describes the branch-and-prune process.

Data:

bound = $[M, +\infty]$, where M is the largest positive floating-point number

X = search domain of the problem

S = a container of branched problems

ϵ = a constant for the desired precision of the result

Output: R = a container of the set of all solutions to the problem

```
1  put X in S;
2  while S is not empty do
3      Y = first element of S;
4      remove Y from S;
5      if underline(f(Y)) < overline(bound) then
6          if overline(f(Y)) < underline(bound) or (underline(f(Y)) < underline(bound) and overline(f(Y)) < overline(bound)) then
7              bound = f(Y);
8              if R is not empty then empty R;
9          else
10             if overline(f(Y)) < overline(bound) then bound = [underline(bound), overline(f(Y))];
11             else if underline(f(Y)) < underline(bound) then bound = [underline(f(Y)), overline(bound)];
12         end
13     end
14     if width(Y) < ε then put Y in R;
15     else
16         split Y into Y1 and Y2;
17         put Y1 and Y2 in S;
18     end
19 end
```

Algorithm 1: Branch-and-bound

Data:

bound $= [M, +\infty]$, where M is the largest positive floating-point number

X = search domain of the problem

S = a container of branched problems

ϵ = a constant for the desired precision of the result

Output: R = a container of the set of all solutions to the problem

```
1  put X in S;
2  while S is not empty do
3      Y = first element of S;
4      remove Y from S;
5      Y' = prune(Y);
6      if Y' is inconsistent then continue;
7      if underline(f(Y')) < overline(bound) then
8          if overline(f(Y')) < underline(bound) or (underline(f(Y')) < underline(bound) and overline(f(Y')) < overline(bound)) then
9              bound = f(Y');
10             if R is not empty then empty R;
11         else
12             if overline(f(Y')) < overline(bound) then bound = [underline(bound), overline(f(Y'))];
13             else if underline(f(Y')) < underline(bound) then bound = [underline(f(Y')), overline(bound)];
14         end
15     end
16     if width(Y') < ε then put Y' in R;
17     else
18         split Y' into Y1 and Y2;
19         put Y1 and Y2 in S;
20     end
21 end
```

Algorithm 2: Branch-and-prune

2.5 Related work: existing interval analysis, constraint solving and optimization libraries

2.5.1 ALIAS-C++

ALIAS is a library for interval analysis, designed to work under Unix or Linux environments. It can solve systems of linear equations and inequalities, optimization and linear algebra problems. The name ALIAS stands for *Algorithms Library of Interval Analysis for equation Systems*, and contains a variety of algorithms based around using interval analysis to help solve different types of problems. It was developed for and is the main development platform for the Constraint solving, OPtimization and Robust interval analysis (COPRIN) project initiated in 2001 at the French institute for research in computer science and automation (INRIA) [33].

2.5.2 IBEX

IBEX is a constraint processing library written in C++ created in 2007 as an open-source project, and includes algorithms to handle non-linear constraints using real numbers with interval arithmetic to account for roundoff errors. It works under a *contractor programming paradigm*, which means that to program a constraint-based solver it is necessary to declare contractor objects that are initialized with the constraints of the problem, and are called upon to contract specific box domains. The library includes multiple contractor algorithms that use different contracting strategies, such as forward-backward, propagation, HC4 and inverse contractor. The main advantage of using contractors is that it is possible to combine multiple contractors in a single algorithm, which makes IBEX a very flexible library. There is also an n extension of the IBEX library called IBEXOPT that uses the contractor model in conjunction with a branch-and-bound framework to do optimization [8].

2.5.3 BARON

BARON (Branch-And-Reduce Optimization Navigator) is a general purpose global solver for non-linear and mixed-integer non-linear programs. At its core, it uses a deterministic branch-and-bound algorithms in conjunction with constraint propagation and duality techniques to find global optima. It is one of the most well-known and award-winning optimizers thanks to its broad number of options and efficiency in finding optima. It was developed partially by the Sahinidis research group at University of Illinois at Urbana-Champaign starting in 2001, and in 2002 it was made available under a commercial software license by The Optimization Firm, who currently continue developing and supporting BARON. Additionally, there are online solving services, such as the NEOS Server run by the University of Wisconsin-Madison, that allow submitting files that contain problems in the GAMS format to their server running BARON [50, 40].

2.5.4 MINOS

MINOS is a general purpose non-linear programming solver created by Bruce Murtagh and Michael Saunders, currently developed by Stanford Business Software, Inc. It is implemented in FORTRAN 77. MINOS works best with problems with continuous objective functions and sparse linear constraints, and problems in which the evaluation of the gradient of the objective function is cheap to evaluate. As is the case with BARON, this solver is offered under a commercial license, but can be accessed in the NEOS server by submitting a file written in the AMPL format to describe the problem [38, 40].

2.5.5 DONLP2

DONLP2 [55] is a solver for nonlinear differentiable optimization problems with equality and inequality constraints. It uses sequential quadratic programming, an iterative method that uses quadratic approximations to the objective function and linear approximations of the constraints to solve the original problem. It was written in Fortran77 by Dr. Peter

Spellucci from the Technical University at Darmstadt. It requires the user to supply functions written in Fortran77 that return the values for the objective function and constraints, as well as their respective gradients.

2.5.6 RealPaver

RealPaver [14] is a software solver for systems of linear systems of equations and problem modeling. RealPaver uses constraint satisfaction techniques with a branch-and-prune algorithm to generate a series of domain boxes whose union encloses the solution to the problem. RealPaver is one of the key components in this thesis, as a domain contractor.

Chapter 3

The speculative algorithm: original implementation and results

Our contribution is an algorithm that we call speculative because it "bets"/speculates on what the expected minimum is and that is implemented using interval and constraint satisfaction techniques. In this chapter, we will describe the original version of this algorithm, report experimental results, and analyze the value of our approach.

3.1 Speculative algorithm

Traditional approaches to global optimization are variations of branch-and-bound [10] / branch-and-prune algorithms [58]. What such algorithms have in common is the exhaustive exploration of the search space (which matters to us for reliable results of global optimization problems) and splitting of the search space as the driving search exploration process. For a full exploration of the search space, and hence splitting on most if not all dimensions, the number of generated sub-problems to be addressed grows exponentially with the dimension of the problem. With our speculative algorithm, we aim to focus the search on one driving dimension instead of the full search space and induce search space reduction through pruning: the one dimension we focus on is the range of the objective function. Speculations are made that "bet" that the expected minimum of the objective function should be in the lower half of the objective function's range.

As a result, our algorithm is a traditional branch-and-prune algorithm, but one that aims to only split (explore) the objective function's range. Speculations are translated into

constraints that are in turn used to contract (prune) the search space. In this regard, using such speculations is in line with previous work on adding redundant constraints (as ways to better contract the search spaces and therefore split less).

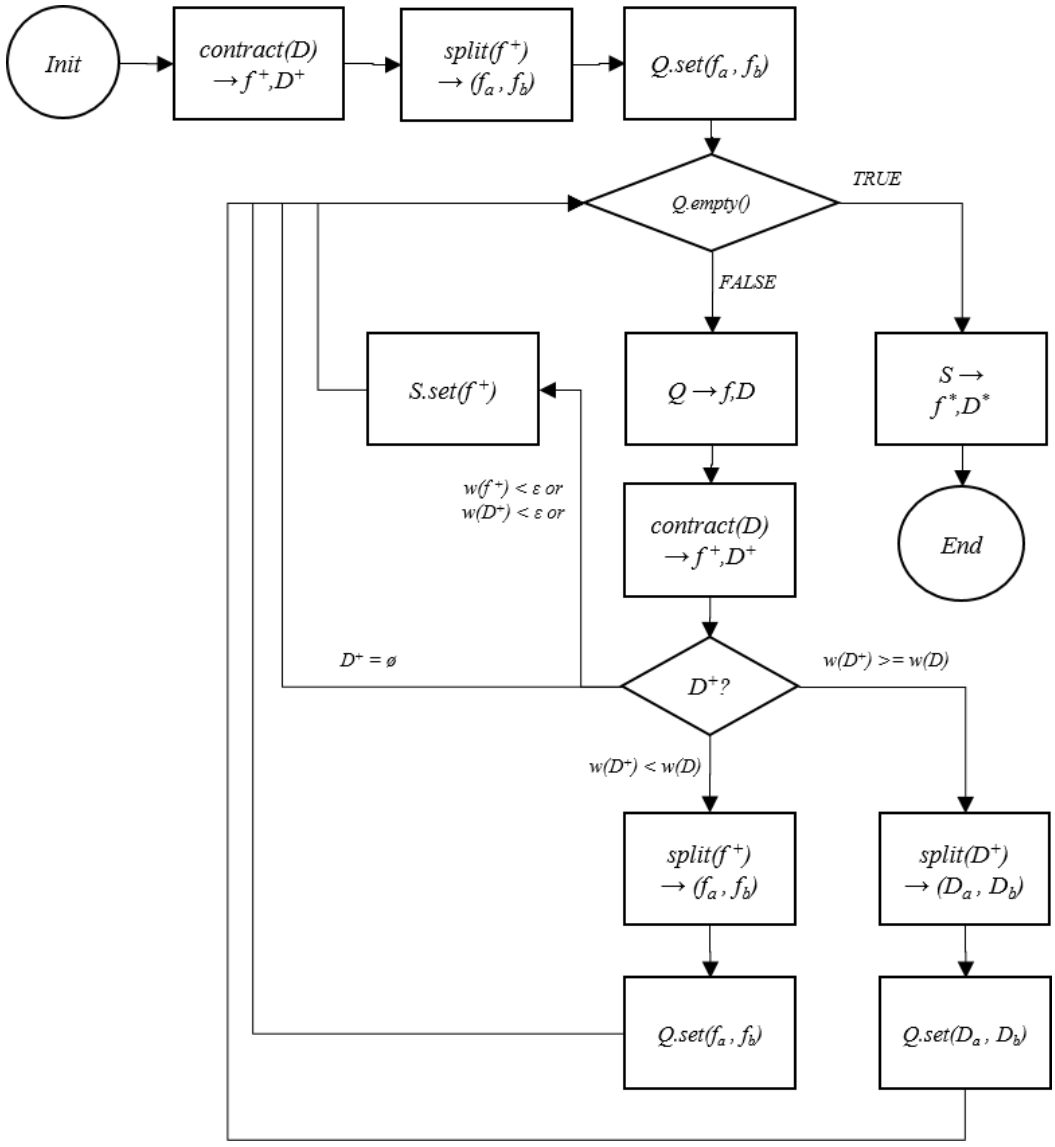

Figure 3.1: Flow chart for the Speculative Algorithm

Figure 3.1 shows the main flow of the algorithm. In general, the main steps of the algorithm are:

1. *Initialize*: set the initial domain, objective function bounds of the problem.

2. *Contract*: reduce the domain using constraint satisfaction techniques. If the domain and objective function combination has no solution, the contraction process fails and the subproblem is discarded.

3. *Split*: create two subproblems based on an initial problem. A successful contraction uses split by objective function, while an unsuccessful contraction uses split by domain.

4. *Get*: obtain the problem with the next lowest objective function value of all generated problems that have not been processed. The algorithm finishes when there are no more subproblems to process.

5. *Store*: store a subproblem for processing.

6. *Validate*: check the width of the objective function and domain of a subproblem. If the widths of both are below a certain threshold, the subproblem is marked as a solution. Otherwise, determines if the contraction is sufficient to continue splitting by objective function or splitting by domain.

Speculation is a branching process that uses an interval objective function. The algorithm obtains the interval value of the objective function using interval arithmetic and interval extensions of functions. This value, $[a, b]$ is bisected into two intervals, $[a, (a+b)/2]$ and $[(a+b)/2, b]$. Each one is incorporated into its own subproblem.

This is when "speculation" occurs: The algorithm, looking for the minimum value, will always select the problems that have lower objective function values. This is the selection criteria for each loop.

3.2 Methodology

To test the performance of the speculative algorithm (Algorithm 1), we need another algorithm to compare against, and a set of test cases to process. The algorithm we use as a

base is the standard branch-and-prune algorithm from Section 2.4.3, our Algorithm 0.

Algorithm 0 splits the *domain* of the problem by bisecting one of the intervals of the domain, creating two new subproblems, which are placed on a priority queue of subproblems. Every time the algorithm selects a subproblem from the queue, it *prunes* the parts of the domain that are inconsistent with the constraints by using *constraint solving* techniques – a *domain contraction*. The algorithm finds a solution when a contracted, consistent box is smaller than a certain precision.

We use test cases from two sources: The COCONUT project benchmarks for global optimization and constraint satisfaction [52], and the Numerica benchmarks [57]. The suite comprises 69 optimization problems, 28 unconstrained and 41 constrained. All tests have a timeout of 120 minutes, after which the algorithm terminates and reports the best solution found, if any, along with log files that trace the search.

Algorithms 0 and 1 are implemented using the Python 2.7 programming language [51], compiled and run on a PC running on an Intel Xeon E5540 Quad-core processor at 2.33 GHz with 12 GB of Memory and the Linux Ubuntu 12.04 64-bit operating system. For domain reduction / contraction, both algorithms use use RealPaver 0.4 [16], a complete interval solver based on constraint satisfaction techniques. RealPaver runs with parameters that prevent solving the test as a constraint satisfaction problem by limiting the algorithm to contraction without domain splitting / bisection, and returning a hull of the domains that contain a potential solution.

3.3 Results

We show the results of running Algorithm 0 and Algorithm 1 in tables that have the following information:

- Benchmark: name of the test case
- n: number of variables

- m: number of constraints
- f_{Bench}: global minimum reported by the benchmark
- f_{Alg0}: upper bound of the best interval minimum obtained by Algorithm 0
- f_{Alg1}: upper bound of the best interval minimum obtained by Algorithm 1
- $(f_{Alg0} - f_{Bench})$: difference between the Algorithm 0 upper bound to the benchmark global minimum
- $(f_{Alg1} - f_{Bench})$: difference between the Algorithm 1 upper bound to the benchmark global minimum
- t_{Alg0}: final execution time of Algorithm 0
- t_{Alg1}: final execution time of Algorithm 1

The results are classified according to how much difference there is between the results of Algorithms 0 and 1. White background is for problems that Algorithm 1 can solve with either an accurate solution (a difference from the benchmark minimum of less than $1.0E-03$), or a quasi-solution (a difference of less than 1). A gray background is for test cases that Algorithm 1 cannot solve, with *italics* for test cases that time out before returning a minimum. A *timeout* result means that the algorithm finished execution by timing out but could not find a minimum. Results displaying *RP Error* failed to complete their execution due to internal errors in RealPaver.

We present the results in four tables. Table 3.1 compares the upper bounds found by Algorithm 0 and Algorithm 1 against the benchmark solutions of unconstrained problems. Table 3.2 compares the execution times of Algorithm 0 and Algorithm 1 on unconstrained problems. Table 3.3 and Table 3.4 show the same results for the constrained problem benchmarks.

3.3.1 Analysis of results

Unconstrained optimization problems

The results on Table 3.1 for Algorithm 1 on unconstrained optimization problems are favorable, accurately solving 12 out of 28 problems, with Algorithm 0 solving 15 problems. The results for the remaining problems are local minima (such as the *levy3* case), problems that do not produce a solution before timing out, and problems that could not be solved due to issues with the external library.

The time comparison on Table 3.2 show that if both Algorithm 0 and Algorithm 1 can solve a problem, Algorithm 1 finds the global minimum faster, with some cases (*levy1, ex4_1_4, ex4_1_7*) showing speedup by an order of magnitude or more.

Constrained optimization problems

Table 3.3 shows the results for Algorithms 0 and 1 on constrained optimization benchmarks. Only 5 out of 41 problems were successfully solved by Algorithm 1, in contrast with the 15 problems solved by Algorithm 0.

Quasi-solutions are local minima found by the algorithm before finding the global minimum after timing out. Algorithm 1 finds quasi-solutions (local minima that are close to the global minimum) in 20 problems, sharing many with the 16 problems in which Algorithm 0 finds quasi-solutions.

It is interesting to note that both algorithms time out in most problems. Algorithm 0 finishes running before timing out on 4 benchmarks, while Algorithm 1 finishes on 2.

3.3.2 Reflection on our proposed approach

The speculative algorithm (Algorithm 1) shows promise on unconstrained optimization. The set of problems solved by Algorithm 1 on unconstrained benchmarks show that it can compete with traditional branch-and-prune. The set of problems that time out before returning any minima are an opportunity to introduce techniques into the unconstrained

optimization process to speed up the speculative process and find global optima faster than Algorithm 0.

Constrained optimization is more challenging. Both Algorithm 0 and Algorithm 1 find solutions and quasi-solutions for more than half of the problems, and Algorithm 0 solves more problems than Algorithm 1. Another important observation is the time it takes for both algorithms to finish running: Only in few cases, either algorithm will complete the search, instead reporting preliminary results upon timing out. Just like in unconstrained optimization, the Algorithm needs techniques that can improve the computation time of the global search.

Table 3.1: Table of results for Algorithms 0 and 1 on unconstrained problems

Benchmark	n	f_{Bench}	f_{Alg0}	f_{Alg1}	$(f_{Alg0} - f_{Bench})$	$(f_{Alg1} - f_{Bench})$
rbrock	2	0	3.9635E-13	6.96604E-08	3.9635E-13	6.96604E-08
ex4_1_6	1	7	7.00000028	7.00000028	2.8E-07	2.8E-07
ex4_1_4	1	0	2.94463E-07	2.94463E-07	2.94463E-07	2.94463E-07
ex4_1_3	1	-443.6717047	-443.6717041	-443.6717044	6.511E-07	3.111E-07
ex4_1_7	1	-7.5	-7.49999961	-7.49999965	3.9E-07	3.5E-07
floudas	1	-7.48731236	-7.48731207	-7.48731199	2.9E-07	3.7E-07
ex4_1_2	1	-663.5000966	-663.5000962	-663.5000962	3.705E-07	4.005E-07
levy1	1	7	7.00000041	7.00000049	4.1E-07	4.9E-07
ex4_1_1	1	-7.487312365	-7.48731207	-7.48731181	2.949E-07	5.549E-07
bqp1var	1	0	6.66134E-16	6.02019E-07	6.66134E-16	6.02019E-07
camel6	2	-1.031628454	-1.03158416	-1.0315888	4.42935E-05	3.96535E-05
more6	2	124.3621	124.3622086	124.3621932	0.00010855	9.324E-05
levy3	2	-186.7309	-20.15315665	-10.31721644	166.5777434	176.4136836
booth	*2*	*0*	*-3.84166963*	*(Timeout)*	*-3.84166963*	*(Timeout)*
cube	*2*	*1.25421E-12*	*8.49321E-13*	*(Timeout)*	*-4.04889E-13*	*(Timeout)*
schwefel1	*3*	*0*	*5.77316E-15*	*(Timeout)*	*5.77316E-15*	*(Timeout)*
hump	*2*	*0*	*8.08713E-12*	*(Timeout)*	*8.08713E-12*	*(Timeout)*
ex8_1_1	*2*	*-2.021806783*	*-2.02180613*	*(Timeout)*	*6.534E-07*	*(Timeout)*
floudas2	*2*	*3*	*4.13749694*	*(Timeout)*	*1.13749694*	*(Timeout)*
beale	*2*	*0*	*14.203125*	*(Timeout)*	*14.203125*	*(Timeout)*
levy5	*2*	*-186.7309*	*(Timeout)*	*(Timeout)*	*(Timeout)*	*(Timeout)*
levy2	*1*	*14.50800792*	*(Timeout)*	*(Timeout)*	*(Timeout)*	*(Timeout)*
eg1	*3*	*-1.429306754*	*(Timeout)*	*(Timeout)*	*(Timeout)*	*(Timeout)*
ex4_1_5	*2*	*4.23865E-13*	*(Timeout)*	*(Timeout)*	*(Timeout)*	*(Timeout)*
ex8_1_2	*1*	*-1.070861019*	*(Timeout)*	*(Timeout)*	*(Timeout)*	*(Timeout)*
denschne	*3*	*1.58063E-15*	*(RP Error)*	*(RP Error)*	*(RP Error)*	*(RP Error)*
ex8_1_4	*2*	*0*	*(RP Error)*	*(RP Error)*	*(RP Error)*	*(RP Error)*
ex8_1_5	*2*	*-1.031628454*	*(RP Error)*	*(RP Error)*	*(RP Error)*	*(RP Error)*

Table 3.2: Table of time comparison for Algorithms 0 and 1 on unconstrained problems

Benchmark	n	$(f_{Alg0} - f_{Bench})$	$(f_{Alg1} - f_{Bench})$	t_{Alg0}	t_{Alg1}
bqp1var	1	6.66134E-16	6.02019E-07	0.010575056	0.086140156
ex4_1_7	1	3.9E-07	3.5E-07	111.5483329	0.873564959
rbrock	2	3.9635E-13	6.96604E-08	0.016638994	4.26500082
ex4_1_1	1	2.949E-07	5.549E-07	8.651198149	6.515843153
floudas	1	2.9E-07	3.7E-07	8.656403065	16.21564603
ex4_1_3	1	6.511E-07	3.111E-07	94.39900398	63.61163116
levy1	1	4.1E-07	4.9E-07	1241.616705	286.2372129
ex4_1_2	1	3.705E-07	4.005E-07	377.966285	302.0987711
ex4_1_4	1	2.94463E-07	2.94463E-07	1765.448433	596.971998
ex4_1_6	1	2.8E-07	2.8E-07	776.702502	606.7931881
camel6	2	4.42935E-05	3.96535E-05	7200	7200
more6	2	0.00010855	9.324E-05	7200	7200
levy3	2	166.5777434	176.4136836	7200	7200
cube	*2*	*-4.04889E-13*	*(Timeout)*	*0.012496948*	*(Timeout)*
schwefel1	*3*	*5.77316E-15*	*(Timeout)*	*0.020755053*	*(Timeout)*
ex8_1_1	*2*	*6.534E-07*	*(Timeout)*	*1.221046925*	*(Timeout)*
beale	*2*	*14.203125*	*(Timeout)*	*7200*	*(Timeout)*
booth	*2*	*-3.84166963*	*(Timeout)*	*7200*	*(Timeout)*
floudas2	*2*	*1.13749694*	*(Timeout)*	*7200*	*(Timeout)*
hump	*2*	*8.08713E-12*	*(Timeout)*	*7200*	*(Timeout)*
levy2	*1*	*-20.74068873*	*(Timeout)*	*7200*	*(Timeout)*
levy5	*2*	*205.9317625*	*(Timeout)*	*7200*	*(Timeout)*
eg1	*3*	*(Timeout)*	*(Timeout)*	*(Timeout)*	*(Timeout)*
ex4_1_5	*2*	*(Timeout)*	*(Timeout)*	*(Timeout)*	*(Timeout)*
ex8_1_2	*1*	*(Timeout)*	*(Timeout)*	*(Timeout)*	*(Timeout)*
ex8_1_4	*2*	*(RP error)*	*(RP error)*	*(RP error)*	*(RP error)*
ex8_1_5	*2*	*(RP error)*	*(RP error)*	*(RP error)*	*(RP error)*
denschne	*3*	*(RP error)*	*(RP error)*	*(RP error)*	*(RP error)*

Table 3.3: Table of results for Algorithms 0 and 1 on constrained problems

Benchmark	n	m	f_{Bench}	f_{Alg0}	f_{Alg1}	$(f_{Alg0} - f_{Bench})$	$(f_{Alg1} - f_{Bench})$
ex14_1_1	3	4	0	1.7763568394002631e-15	9.8975542928937667e-14	1.77636E-15	9.89755E-14
h79	5	3	0.078776821	0.0787772734945075	0.0787787965794259	4.52595E-07	1.97568E-06
h80	5	3	0.053949504	0.0539507019961389	0.0539523074802214	1.198E-06	2.80348E-06
ex6_2_14	4	2	-0.695357935	-0.6953542331442246	-0.6953536877850460	3.70146E-06	4.24681E-06
h78	5	3	-2.919705926	-2.9196870960229289	-2.9196580695402128	1.88301E-05	4.78566E-05
ex7_2_6	3	1	-83.24993533	-83.2497167018888149	-83.2497206503360445	0.000218624	0.000214676
h77	5	2	0.241505129	0.2415204736930432	0.2418813683213293	1.53449E-05	0.00037624
h76	4	3	-4.681818182	-4.6817870844290361	-4.6811920949519426	3.10974E-05	0.000626087
ex6_1_4	6	4	-0.294541281	-0.2945310825059501	-0.2919864399170705	1.01983E-05	0.002554841
h75	4	4	5174.41	5174.4126958348879270	5174.4126958853266842	0.002695835	0.002695885
h81	5	3	0.05394951	0.0539504036213946	0.0567439971587578	8.93921E-07	0.002794487
h95	6	4	0.0156195	0.0232702862022400	0.0232706135762388	0.007650786	0.007651114
h96	6	4	0.0156195	0.0232702862022400	0.0232706135762388	0.007650786	0.007651114
ex6_1_1	8	6	-0.020235863	-0.0088160553191247	-0.0088161537187099	0.011419808	0.011419709
chance	4	3	29.8944	29.8995337661	29.9094896334011295	0.005133766	0.015089633
h73	4	3	29.8944	29.8980398342274434	29.9150296115732814	0.003639834	0.020629612
ex2_1_9	10	1	-0.375	-0.3591307624818594	-0.3233904269732101	0.015869238	0.051609573
ex6_2_8	3	1	-0.027006349	0.0441786372486723	0.0441786871456789	0.071184986	0.071185036
ex6_2_13	6	3	-0.216209674	-0.1303099938849697	-0.1303099484693406	0.08589968	0.085899726
ex6_2_6	3	1	7.10763E-07	0.0898119939931157	0.0898119940561275	0.089811283	0.089811283
ex6_2_12	4	2	0.289194749	0.3935396548003712	0.3935396563260850	0.104344906	0.104344908
h72	4	6	727.5418231	727.6806285798676299	727.6836603055073738	0.138805507	0.141837233
ex6_2_9	4	2	-0.034066184	0.1562764315232283	0.1562764305484650	0.190342616	0.190342615
ex6_2_11	3	1	1.5569E-06	0.2830582241230852	0.2830581386124653	0.283056667	0.283056582
ex6_2_10	6	3	-3.051976126	-2.7272043821840164	-2.7272043915747144	0.324771744	0.324771734
O32	5	7	-30665.53867	-30665.5386710330567439	-30653.4728249678082648	-1.033E-06	12.06584503
h113	10	8	24.30620906	42.7521582379402076	41.4562999991938668	18.44594918	17.15009094
ex2_1_8	24	10	15639	40720.5423036566935480	40698.8092548310814891	25081.5423	25059.80925
ex14_2_6	*5*	*7*	*0*	*2.2204460492503471e-16*	*(Timeout)*	*2.22045E-16*	*(Timeout)*
ex14_2_1	*5*	*7*	*0*	*2.2204460492503668e-16*	*(Timeout)*	*2.22045E-16*	*(Timeout)*
ex14_2_3	*6*	*9*	*0*	*2.2204460492503668e-16*	*(Timeout)*	*2.22045E-16*	*(Timeout)*
ex14_2_7	*6*	*9*	*0*	*2.2204460492503668e-16*	*(Timeout)*	*2.22045E-16*	*(Timeout)*
ex6_1_3	*12*	*9*	*-0.352510346*	*-0.3243435684570191*	*(Timeout)*	*0.028166778*	*(Timeout)*
h98	*6*	*4*	*3.13581*	*4.3768060070896846*	*(Timeout)*	*1.240996007*	*(Timeout)*
h97	*6*	*4*	*3.135809123*	*4.9241399697866237*	*(Timeout)*	*1.788330847*	*(Timeout)*
ex3_1_1	*8*	*6*	*7049.208345*	*7060.1760864258794754*	*(Timeout)*	*10.96774122*	*(Timeout)*
h106	*8*	*14*	*7049.20834*	*7104.0527343750436557*	*(Timeout)*	*54.84439458*	*(Timeout)*
ex7_2_1	*7*	*14*	*1227.189572*	*1487.1522906643435817*	*(Timeout)*	*259.9627183*	*(Timeout)*
ex2_1_7	*20*	*10*	*-4150.410134*	*-1088.4375*	*(Timeout)*	*3061.972634*	*(Timeout)*
ex7_3_4	*12*	*17*	*6.274634336*	*1.00E+17*	*(Timeout)*	*1.00E+17*	*(Timeout)*
ex7_3_5	*13*	*15*	*1.203630389*	*(Timeout)*	*(Timeout)*	*(Timeout)*	*(Timeout)*

Table 3.4: Table of time comparison for Algorithms 0 and 1 on constrained problems

Benchmark	n	m	$(f_{Alg0} - f_{Bench})$	$(f_{Alg1} - f_{Bench})$	t_{Alg0}	t_{Alg1}
ex14_1_1	3	4	1.77636E-15	9.89755E-14	2.125368118	2.289428949
h75	4	4	0.002695835	0.002695885	6.470372915	10.18295598
h79	5	3	4.52595E-07	1.97568E-06	7200	7200
h80	5	3	1.198E-06	2.80348E-06	7200	7200
ex6_2_14	4	2	3.70146E-06	4.24681E-06	7200	7200
h78	5	3	1.88301E-05	4.78566E-05	7200	7200
ex7_2_6	3	1	0.000218624	0.000214676	7200	7200
h77	5	2	1.53449E-05	0.00037624	7200	7200
h76	4	3	3.10974E-05	0.000626087	7200	7200
ex6_1_4	6	4	1.01983E-05	0.002554841	7200	7200
h81	5	3	8.93921E-07	0.002794487	7200	7200
h95	6	4	0.007650786	0.007651114	7200	7200
h96	6	4	0.007650786	0.007651114	7200	7200
ex6_1_1	8	6	0.011419808	0.011419709	7200	7200
chance	4	3	0.005133766	0.015089633	7200	7200
h73	4	3	0.003639834	0.020629612	7200	7200
ex2_1_9	10	1	0.015869238	0.051609573	7200	7200
ex6_2_8	3	1	0.071184986	0.071185036	7200	7200
ex6_2_13	6	3	0.08589968	0.085899726	7200	7200
ex6_2_6	3	1	0.089811283	0.089811283	7200	7200
ex6_2_12	4	2	0.104344906	0.104344908	7200	7200
h72	4	6	0.138805507	0.141837233	7200	7200
ex6_2_9	4	2	0.190342616	0.190342615	7200	7200
ex6_2_11	3	1	0.283056667	0.283056582	7200	7200
ex6_2_10	6	3	0.324771744	0.324771734	3	7200
O32	5	7	-1.033E-06	12.06584503	415.4259429	7200
h113	10	8	18.44594918	17.15009094	7200	7200
ex2_1_8	24	10	25081.5423	25059.80925	7200	7200
ex14_2_6	*5*	*7*	*2.22045E-16*	*(Timeout)*	*7200*	*(Timeout)*
ex14_2_1	*5*	*7*	*2.22045E-16*	*(Timeout)*	*7200*	*(Timeout)*
ex14_2_3	*6*	*9*	*2.22045E-16*	*(Timeout)*	*7200*	*(Timeout)*
ex14_2_7	*6*	*9*	*2.22045E-16*	*(Timeout)*	*7200*	*(Timeout)*
ex6_1_3	*12*	*9*	*0.028166778*	*(Timeout)*	*7200*	*(Timeout)*
h98	*6*	*4*	*1.240996007*	*(Timeout)*	*7200*	*(Timeout)*
h97	*6*	*4*	*1.788330847*	*(Timeout)*	*7200*	*(Timeout)*
ex3_1_1	*8*	*6*	*10.96774122*	*(Timeout)*	*7200*	*(Timeout)*
h106	*8*	*14*	*54.84439458*	*(Timeout)*	*7200*	*(Timeout)*
ex7_2_1	*7*	*14*	*259.9627183*	*(Timeout)*	*7200*	*(Timeout)*
ex2_1_7	*20*	*10*	*3061.972634*	*(Timeout)*	*7200*	*(Timeout)*
ex7_3_4	*12*	*17*	*1E+17*	*(Timeout)*	*7200*	*(Timeout)*
ex7_3_5	*13*	*15*	*(Timeout)*	*(Timeout)*	*(Timeout)*	*(Timeout)*

Chapter 4

Improvements on the speculative algorithm

We want to improve the performance of the speculative algorithm (Algorithm 1). We propose incorporating different techniques for unconstrained and constrained problems, to take advantage of specific properties in them. For unconstrained problems, we add two improvements: use local search to improve the global upper bound with local minima, and analyze the monotonicity of the function on a given domain to find and improve existing domain bounds. For constrained problems, we create a linear problem using the Karush-Kuhn-Tucker optimality conditions and solve it using interval Newton method to obtain an improved upper bound based on the optimality conditions. The speculative algorithm that incorporates these improvements is our Algorithm 2.

4.1 Unconstrained optimization

The basic speculative algorithm is a derivative-free algorithm, which means it uses heuristics to guide the search. For unconstrained problems, approaching the solution using and splitting fine-grained interval subdomains and speculated objective function values can result in many finely-grained contiguous subdomains surrounding a solution. Processing these is extremely time consuming, as RealPaver struggles with attempting to contract fine-grained domains and objective functions. For those cases, we switch to a local search algorithm. We use the Nelder-Mead Simplex method [42], as implemented in the SciPy package [26] for Python.

In the original algorithm, the algorithm selects an interval variable for domain splitting based on the widest interval variable in the domain. This naive approach does not guarantee that either half of the domain will provide a domain contraction that will improve the time performance of the algorithm. We propose an approach that analyzes the monotonicity of the objective function, using partial derivatives. Partial derivatives inform the algorithm on the behavior of the function to make better bisections and domain contraction.

If a domain D has n variables, then for the objective funtion F has n partial derivatives F_i'. The interval evaluation of these derivatives results in interval values f_i', describing the slope on each dimension of the domain. An interval derivative value containing zero means that a minimum for that variable is found in that domain, with no way of knowing at which value. These intervals are candidates for splitting. Interval derivative values that do not contain zero imply that the dimension of the value of the domain corresponding to that partial derivative is *monotonic*, that is the function is sloped exclusively up or down. A monotonic dimension on the domain can be reduced implicitly to the domain bounds closest to the minimum. In those cases, we set the value of the domain to the edge in the direction that the derivative points out: A negative slope uses the upper bound of that specific variable interval, and a positive slope uses the lower bound. The subprocess in Figure 4.1 shows how this process takes place in Algorithm 2.

4.2 Constrained optimization

For unconstrained problems, Algorithm 2 uses local search to improve the value of the upper bound of the objective function. This process is straightforward. For constrained optimization, the presence of constraints introduces complications that do not exist in unconstrained problems. To apply the same principle of using local search to improve global search, we propose using a two-part technique for constrained problems in Algorithm 2: First, create a linear problem based on the constrained problem using the Karush-Kuhn-Tucker conditions for optimality; then, solve this new problem using interval Newton

$F' \leftarrow obtainPartialDerivatives(F)$

$n \leftarrow getNumberOfVariables(D)$

$split \leftarrow NULL$

$maxWidth \leftarrow 0$

$X \leftarrow getBox(D)$

for $i = 0$ to n **do**

  if $0 \in F_i'(X)$ **then**

    if $width(F_i'(X)) > maxWidth$ **then**

      $maxWidth \leftarrow width(F_i'(X))$

      $split \leftarrow i$

    end if

  else if $\overline{F_i'(X)} < 0$ **then**

    $X_i \leftarrow [\lfloor \overline{X_i} \rfloor, \lceil \overline{X_i} \rceil]$

  else

    $X_i \leftarrow [\lfloor \underline{X_i} \rfloor, \lceil \underline{X_i} \rceil]$

  end if

end for

$(D_a, D_b) \leftarrow splitOn(X, split)$

Figure 4.1: Domain contraction and splitting based on monotonocity analysis of the objective function

method to obtain a domain that the algorithm evaluates to produce a new upper bound on the objective function for the original constrained optimization problem.

4.2.1 Karush-Kuhn-Tucker conditions

Algorithm 2 creates a linear problem that can be solved with a local method through the Karush-Kuhn-Tucker (KKT) conditions [31], a series of necessary conditions that a point must have to be considered a solution to a constrained optimization problem. The conditions use the derivatives of the objective function and the constraints to build a system of linear equations. They can test the optimizality of a point, as well as solving the system of equations created by the KKT conditions to find a point that satisfies them and is also a minimum [6].

Definition 23 For an optimization problem

$$\begin{aligned} &\text{minimize } f(X) \\ &\text{s.t. } g_i(X) \leq 0, \quad i = 1, \ldots, m \\ &h_k(X) = 0, \quad k = 1, \ldots, p \end{aligned}$$

we say that the point x^* is a minimum solution to the problem, if it satisfies the following conditions:

Stationarity

$$\nabla f(x^*) = -\sum_{i=1}^{m} \mu_i \nabla g_i(x^*) - \sum_{j=1}^{l} \lambda_j \nabla h_j(x^*)$$

Primal feasibility

$$\begin{aligned} g_i(x^*) \leq 0, \text{ for all } i = 1, \ldots, m \\ h_j(x^*) = 0, \text{ for all } j = 1, \ldots, l \end{aligned}$$

Dual feasibility

$$\mu_i \geq 0, \text{ for all } i = 1, \ldots, m$$

Complimentary slackness

$$\mu_i g_i(x^*) = 0, \text{for all } i = 1, \ldots, m$$

Algorithm 2 uses KKT conditions to create a linear problem from the constrained optimization problem, then solves this linear problem using interval Newton method from Section 2.3.1. Interval Newton is part of the algorithms in RealPaver [14]. Algorithm 2 uses the interval Newton method in RealPaver to solve the linear problem of the KKT conditions.

Incorporating the KKT conditions

For constrained problems, Algorithm 2 follows the same process described for Algorithm 1 (Section 3.1), with two modifications. The first modification is a *preconditioning*. Before the main loop of the speculative process, Algorithm 2 creates the KKT linear problem solves it with RealPaver's interval Newton method on the initial domain. Interval Newton returns a new domain, which is used to evaluate the original objective function. Algorithm 2 uses the upper bound of this evaluation and the new upper bound of the *original* objective function, then splits the objective function to begin the speculative search.

The second modification introduces the KKT conditions into the iterative process. After the contraction at the beginning of the loop, Algorithm 2 verifies whether RealPaver was able to contract the domain. A contraction results in continued speculation. No contraction means we need to find a new way to bound the objective function. Algorithm 2 uses the same steps as the preconditioning (create KKT linear problem, solve with interval Newton, evaluate domain and set new objective function upper bound) to create a new interval objective function that is speculated upon. After setting the new upper-bound, the objective function value is split into two new subproblems that are placed in the queue of subproblems.

4.3 Results

We show the results of running Algorithm 0 and Algorithm 2 in tables that have the following information:

- Benchmark: name of the test case
- n: number of variables
- m: number of constraints
- f_{Bench}: global minimum reported by the benchmark
- f_{Alg0}: upper bound of the best interval minimum obtained by Algorithm 0
- f_{Alg2}: upper bound of the best interval minimum obtained by Algorithm 2
- $(f_{Alg0} - f_{Bench})$: difference between the Algorithm 0 upper bound to the benchmark global minimum
- $(f_{Alg2} - f_{Bench})$: difference between the Algorithm 2 upper bound to the benchmark global minimum
- t_{Alg0}: final execution time of Algorithm 0
- t_{Alg2}: final execution time of Algorithm 2

The results are classified as described in Section 3.3. White background is for problems that Algorithm 1 can solve with either an accurate solution (a difference from the benchmark minimum of less than $1.0E-03$), or a quasi-solution (a difference of less than 1). A gray background is for test cases that Algorithm 2 cannot solve, with *italics* for test cases that time out before returning a minimum. A *timeout* result means that the algorithm finished execution by timing out but could not find a minimum. Results displaying *RP Error* failed to complete their execution due to internal errors in RealPaver.

We present the results in four tables. Table 4.1 compares the upper bounds found by Algorithm 0 and Algorithm 2 against the benchmark solutions of unconstrained problems. Table 4.2 compares the execution times of Algorithm 0 and Algorithm 2 on unconstrained problems. Table 4.3 and Table 4.4 show the same results for the constrained problem benchmarks.

4.3.1 Analysis of results

Unconstrained optimization problems

Table 4.1 shows that Algorithm 2 solves 27 out of 28 problems, including 6 problems that Algorithm 0 is not able to solve, plus 12 problems that Algorithm 0 already solves. This is an improvement from Algorithm 1, which finds solutions for only 12 problems.

The time comparison on Table 4.2 shows that Algorithm 2 solves the 27 problems without reaching the timeout, with many cases finishing considerably faster than Algorithm 0.

Constrained optimization problems

The results for Algorithm 0 and Algorithm 2 on constrained optimizatoin benchmarks are on Table 4.3. Algorithm 2 successfully solves 5 problems, and finds quasi-solutions in 6 others. Algorithm 0 solves 15 problems, and finds quasi-solutions in 16 problems.

Table 4.3 shows the time results for Algorithm 0 and Algorithm 2. For most problems, both algorithms still time out; however, Algorithm 2 finishes running before timing out in 4 benchmarks, in comparison with the 2 finished tests in Algorithm 1.

4.3.2 Reflection on our proposed approach

For unconstrained optimization, Algorithm 2 shows improvement over the results of Algorithm 1. Algorithm 2 solves more problems, and finds the global solution faster than the

baseline Algorithm 0. Additionally, it solves other problems that Algorithm 0 and Algorithm 1 could not solve. We conclude that this technique can speed up the unconstrained optimization process in many cases.

In general, Algorithm 2 does not show any significant improvement over Algorithm 1 or Algorithm 0. We conclude that, in general, solving the KKT conditions to obtain an upper bound of the objective function provides no significant improvement over a speculative process, and is meaningful only in specific problems.

Table 4.1: Table of results for Algorithms 0 and 2 on unconstrained problems

Benchmark	n	f_{Bench}	f_{Alg0}	f_{Alg2}	$(f_{Alg0} - f_{Bench})$	$(f_{Alg2} - f_{Bench})$
ex4_1_5	2	4.23865E-13	(Timeout)	5.98065E-08	(Timeout)	5.98061E-08
eg1	3	-1.429306754	(Timeout)	-1.42930377	(Timeout)	2.9836E-06
ex8_1_2	1	-1.070861019	(Timeout)	11.21335545	(Timeout)	12.28421647
denschne	3	1.58063E-15	(RP Error)	4.68718E-15	(RP Error)	3.10655E-15
ex8_1_4	2	0	(RP Error)	1.42536E-13	(RP Error)	1.42536E-13
ex8_1_5	2	-1.031628454	(RP Error)	-1.03162351	(RP Error)	4.9435E-06
cube	2	1.25421E-12	8.49321E-13	2.67463E-09	-4.04889E-13	2.67338E-09
ex8_1_1	2	-2.021806783	-2.02180613	-2.02180677	6.534E-07	1.34E-08
hump	2	0	8.08713E-12	6.19709E-07	8.08713E-12	6.19709E-07
bqp1var	1	0	6.66134E-16	7.15256E-07	6.66134E-16	7.15256E-07
ex4_1_4	1	0	2.94463E-07	7.85521E-07	2.94463E-07	7.85521E-07
schwefel1	3	0	5.77316E-15	9.92689E-07	5.77316E-15	9.92689E-07
camel6	2	-1.031628454	-1.03158416	-1.03162691	4.42935E-05	1.5435E-06
ex4_1_3	1	-443.6717047	-443.6717041	-443.6717017	6.511E-07	3.0411E-06
ex4_1_7	1	-7.5	-7.49999961	-7.49999208	3.9E-07	7.92E-06
ex4_1_1	1	-7.487312365	-7.48731207	-7.48728657	2.949E-07	2.57949E-05
floudas	1	-7.48731236	-7.48731207	-7.48727432	2.9E-07	3.804E-05
rbrock	2	0	3.9635E-13	5.0091E-05	3.9635E-13	5.0091E-05
ex4_1_6	1	7	7.00000028	7.00084697	2.8E-07	0.00084697
levy1	1	7	7.00000041	7.00102615	4.1E-07	0.00102615
more6	2	124.3621	124.3622086	124.363916	0.00010855	0.00181595
ex4_1_2	1	-663.5000966	-663.5000962	-663.4978341	3.705E-07	0.00226247
floudas2	*2*	*3*	*4.13749694*	*4.13749695*	*1.13749694*	*1.13749695*
beale	*2*	*0*	*14.203125*	*14.203125*	*14.203125*	*14.203125*
levy5	*2*	*-186.7309*	*19.20086245*	*66.61116919*	*205.9317625*	*253.3420692*
levy2	*1*	*14.50800792*	*-6.23268081*	*440.845397*	*-20.74068873*	*426.3373891*
levy3	*2*	*-186.7309*	*-20.15315665*	*214423.8078*	*166.5777434*	*214610.5387*
booth	*2*	*0*	*-3.84166963*	*(Timeout)*	*-3.84166963*	*(Timeout)*

Table 4.2: Table of time comparison for Algorithms 0 and 2 on unconstrained problems

Benchmark	n	$(f_{Alg0} - f_{Bench})$	$(f_{Alg2} - f_{Bench})$	t_{Alg0}	t_{Alg1}
ex4_1_5	2	(Timeout)	5.98061E-08	(Timeout)	3.361732006
ex8_1_2	1	(Timeout)	12.28421647	(Timeout)	324.453711
eg1	3	(Timeout)	2.9836E-06	(Timeout)	1257.215137
ex8_1_4	2	(RP Error)	1.42536E-13	(RP Error)	1.854950905
denschne	3	(RP Error)	3.10655E-15	(RP Error)	2.169813871
ex8_1_5	2	(RP Error)	4.9435E-06	(RP Error)	16.69612598
bqp1var	1	6.66134E-16	7.15256E-07	0.010575056	0.12073493
cube	2	-4.04889E-13	2.67338E-09	0.012496948	0.457427979
rbrock	2	3.9635E-13	5.0091E-05	0.016638994	0.494719982
ex8_1_1	2	6.534E-07	1.34E-08	1.221046925	0.742885113
schwefel1	3	5.77316E-15	9.92689E-07	0.020755053	0.976428986
ex4_1_3	1	6.511E-07	3.0411E-06	94.39900398	4.818313837
hump	2	8.08713E-12	6.19709E-07	7200	6.50304389
ex4_1_4	1	2.94463E-07	7.85521E-07	1765.448433	9.942773104
floudas	1	2.9E-07	3.804E-05	8.656403065	31.3725028
ex4_1_1	1	2.949E-07	2.57949E-05	8.651198149	35.28808188
camel6	2	4.42935E-05	1.5435E-06	7200	36.95381498
ex4_1_7	1	3.9E-07	7.92E-06	111.5483329	108.28582
ex4_1_2	1	3.705E-07	0.00226247	377.966285	13.85145903
levy1	1	4.1E-07	0.00102615	1241.616705	38.55503893
ex4_1_6	1	2.8E-07	0.00084697	776.702502	69.46248388
more6	2	0.00010855	0.00181595	7200	89.4569118
levy5	*2*	*205.9317625*	*253.3420692*	*7200*	*0.677630186*
beale	*2*	*14.203125*	*14.203125*	*7200*	*5.828889132*
levy2	*1*	*-20.74068873*	*426.3373891*	*7200*	*10.27432704*
floudas2	*2*	*1.13749694*	*1.13749695*	*7200*	*20.51519108*
levy3	*2*	*166.5777434*	*214610.5387*	*7200*	*72.82514596*
booth	*2*	*-3.84166963*	*(Timeout)*	*7200*	*(Timeout)*

Table 4.3: Table of results for Algorithms 0 and 2 on constrained problems

Benchmark	n	m	f_{Bench}	f_{Alg0}	f_{Alg2}	$(f_{Alg0} - f_{Bench})$	$(f_{Alg2} - f_{Bench})$
O32	5	7	-30665.53867	-30665.5386710330567439	-30665.5386714601700078	-1.033E-06	-1.4601E-06
ex14_1_1	3	4	0	1.7763568394002631e-15	9.8975542928937667e-14	1.77636E-15	9.89755E-14
h95	6	4	0.0156195	0.0232702862022400	0.0156196202397999	0.007650786	1.2024E-07
h96	6	4	0.0156195	0.0232702862022400	0.0156196202397999	0.007650786	1.2024E-07
h79	5	3	0.078776821	0.0787772734945075	0.0787780411823789	4.52595E-07	1.22028E-06
h77	5	2	0.241505129	0.2415204736930432	0.2418811443556964	1.53449E-05	0.000376016
h73	4	3	29.8944	29.8980398342274434	29.8947846390976615	0.003639834	0.000384639
h76	4	3	-4.681818182	-4.6817870844290361	-4.6811920894192394	3.10974E-05	0.000626092
h75	4	4	5174.41	5174.4126958348879270	5174.4126957747703273	0.002695835	0.002695775
ex2_1_9	10	1	-0.375	-0.3591307624818594	-0.2968748509883791	0.015869238	0.078125149
ex6_2_14	4	2	-0.695357935	-0.6953542331442246	-0.1426058315903528	3.70146E-06	0.552752103
ex14_2_6	*5*	*7*	*0*	*2.2204460492503471e-16*	*(Timeout)*	*2.22045E-16*	*(Timeout)*
ex14_2_1	*5*	*7*	*0*	*2.2204460492503668e-16*	*(Timeout)*	*2.22045E-16*	*(Timeout)*
ex14_2_3	*6*	*9*	*0*	*2.2204460492503668e-16*	*(Timeout)*	*2.22045E-16*	*(Timeout)*
ex14_2_7	*6*	*9*	*0*	*2.2204460492503668e-16*	*(Timeout)*	*2.22045E-16*	*(Timeout)*
h81	*5*	*3*	*0.05394951*	*0.0539504036213946*	*(Timeout)*	*8.93921E-07*	*(Timeout)*
h80	*5*	*3*	*0.053949504*	*0.0539507019961389*	*(Timeout)*	*1.198E-06*	*(Timeout)*
ex6_1_4	*6*	*4*	*-0.294541281*	*-0.2945310825059501*	*(Timeout)*	*1.01983E-05*	*(Timeout)*
h78	*5*	*3*	*-2.919705926*	*-2.9196870960229289*	*(Timeout)*	*1.88301E-05*	*(Timeout)*
ex7_2_6	*3*	*1*	*-83.24993533*	*-83.2497167018888149*	*(Timeout)*	*0.000218624*	*(Timeout)*
chance	*4*	*3*	*29.8944*	*29.8995337661*	*(Timeout)*	*0.005133766*	*(Timeout)*
ex6_1_1	*8*	*6*	*-0.020235863*	*-0.0088160553191247*	*(Timeout)*	*0.011419808*	*(Timeout)*
ex6_1_3	*12*	*9*	*-0.352510346*	*-0.3243435684570191*	*(Timeout)*	*0.028166778*	*(Timeout)*
ex6_2_8	*3*	*1*	*-0.027006349*	*0.0441786372486723*	*(Timeout)*	*0.071184986*	*(Timeout)*
ex6_2_13	*6*	*3*	*-0.216209674*	*-0.1303099938849697*	*(Timeout)*	*0.08589968*	*(Timeout)*
ex6_2_6	*3*	*1*	*7.10763E-07*	*0.0898119939931157*	*(Timeout)*	*0.089811283*	*(Timeout)*
ex6_2_12	*4*	*2*	*0.289194749*	*0.3935396548003712*	*(Timeout)*	*0.104344906*	*(Timeout)*
h72	*4*	*6*	*727.5418231*	*727.6806285798676299*	*(Timeout)*	*0.138805507*	*(Timeout)*
ex6_2_9	*4*	*2*	*-0.034066184*	*0.1562764315232283*	*(Timeout)*	*0.190342616*	*(Timeout)*
ex6_2_11	*3*	*1*	*1.5569E-06*	*0.2830582241230852*	*(Timeout)*	*0.283056667*	*(Timeout)*
ex6_2_10	*6*	*3*	*-3.051976126*	*-2.7272043821840164*	*(Timeout)*	*0.324771744*	*(Timeout)*
h98	*6*	*4*	*3.13581*	*4.3768060070896846*	*(Timeout)*	*1.240996007*	*(Timeout)*
h97	*6*	*4*	*3.135809123*	*4.9241399697866237*	*(Timeout)*	*1.788330847*	*(Timeout)*
ex3_1_1	*8*	*6*	*7049.208345*	*7060.1760864258794754*	*(Timeout)*	*10.96774122*	*(Timeout)*
h113	*10*	*8*	*24.30620906*	*42.7521582379402076*	*(Timeout)*	*18.44594918*	*(Timeout)*
h106	*8*	*14*	*7049.20834*	*7104.0527343750436557*	*(Timeout)*	*54.84439458*	*(Timeout)*
ex7_2_1	*7*	*14*	*1227.189572*	*1487.1522906643435817*	*(Timeout)*	*259.9627183*	*(Timeout)*
ex2_1_7	*20*	*10*	*-4150.410134*	*-1088.4375*	*(Timeout)*	*3061.972634*	*(Timeout)*
ex2_1_8	*24*	*10*	*15639*	*40720.5423036566935480*	*(Timeout)*	*25081.5423*	*(Timeout)*
ex7_3_4	*12*	*17*	*6.274634336*	*1.00E+17*	*(Timeout)*	*1.00E+17*	*(Timeout)*
ex7_3_5	*13*	*15*	*1.203630389*	*(Timeout)*	*(Timeout)*	*(Timeout)*	*(Timeout)*

Table 4.4: Table of time comparison for Algorithms 0 and 2 on constrained problems

Benchmark	n	m	$(f_{Alg0} - f_{Bench})$	$(f_{Alg2} - f_{Bench})$	t_{Alg0}	t_{Alg1}
ex14_1_1	3	4	1.77636E-15	9.89755E-14	2.125368118	2.165382862
O32	5	7	-1.033E-06	-1.4601E-06	415.4259429	13.74955583
h95	6	4	0.007650786	1.2024E-07	7200	15.330621
h96	6	4	0.007650786	1.2024E-07	7200	15.54575491
h79	5	3	4.52595E-07	1.22028E-06	7200	7200
h75	4	4	0.002695835	0.002695775	6.470372915	7.954741001
h77	5	2	1.53449E-05	0.000376016	7200	7200
h73	4	3	0.003639834	0.000384639	7200	7200
h76	4	3	3.10974E-05	0.000626092	7200	7200
ex2_1_9	10	1	0.015869238	0.078125149	7200	7200
ex6_2_14	4	2	3.70146E-06	0.552752103	7200	7200
ex14_2_6	*5*	*7*	*2.22045E-16*	*(Timeout)*	*7200*	*(Timeout)*
ex14_2_1	*5*	*7*	*2.22045E-16*	*(Timeout)*	*7200*	*(Timeout)*
ex14_2_3	*6*	*9*	*2.22045E-16*	*(Timeout)*	*7200*	*(Timeout)*
ex14_2_7	*6*	*9*	*2.22045E-16*	*(Timeout)*	*7200*	*(Timeout)*
h81	*5*	*3*	*8.93921E-07*	*(Timeout)*	*7200*	*(Timeout)*
h80	*5*	*3*	*1.198E-06*	*(Timeout)*	*7200*	*(Timeout)*
ex6_1_4	*6*	*4*	*1.01983E-05*	*(Timeout)*	*7200*	*(Timeout)*
h78	*5*	*3*	*1.88301E-05*	*(Timeout)*	*7200*	*(Timeout)*
ex7_2_6	*3*	*1*	*0.000218624*	*(Timeout)*	*7200*	*(Timeout)*
chance	*4*	*3*	*0.005133766*	*(Timeout)*	*7200*	*(Timeout)*
ex6_1_1	*8*	*6*	*0.011419808*	*(Timeout)*	*7200*	*(Timeout)*
ex6_1_3	*12*	*9*	*0.028166778*	*(Timeout)*	*7200*	*(Timeout)*
ex6_2_8	*3*	*1*	*0.071184986*	*(Timeout)*	*7200*	*(Timeout)*
ex6_2_13	*6*	*3*	*0.08589968*	*(Timeout)*	*7200*	*(Timeout)*
ex6_2_6	*3*	*1*	*0.089811283*	*(Timeout)*	*7200*	*(Timeout)*
ex6_2_12	*4*	*2*	*0.104344906*	*(Timeout)*	*7200*	*(Timeout)*
h72	*4*	*6*	*0.138805507*	*(Timeout)*	*7200*	*(Timeout)*
ex6_2_9	*4*	*2*	*0.190342616*	*(Timeout)*	*7200*	*(Timeout)*
ex6_2_11	*3*	*1*	*0.283056667*	*(Timeout)*	*7200*	*(Timeout)*
ex6_2_10	*6*	*3*	*0.324771744*	*(Timeout)*	*3*	*(Timeout)*
h98	*6*	*4*	*1.240996007*	*(Timeout)*	*7200*	*(Timeout)*
h97	*6*	*4*	*1.788330847*	*(Timeout)*	*7200*	*(Timeout)*
ex3_1_1	*8*	*6*	*10.96774122*	*(Timeout)*	*7200*	*(Timeout)*
h113	*10*	*8*	*18.44594918*	*(Timeout)*	*7200*	*(Timeout)*
h106	*8*	*14*	*54.84439458*	*(Timeout)*	*7200*	*(Timeout)*
ex7_2_1	*7*	*14*	*259.9627183*	*(Timeout)*	*7200*	*(Timeout)*
ex2_1_7	*20*	*10*	*3061.972634*	*(Timeout)*	*7200*	*(Timeout)*
ex2_1_8	*24*	*10*	*25081.5423*	*(Timeout)*	*7200*	*(Timeout)*
ex7_3_4	*12*	*17*	*1E+17*	*(Timeout)*	*7200*	*(Timeout)*
ex7_3_5	*13*	*15*	*(Timeout)*	*(Timeout)*	*(Timeout)*	*(Timeout)*

Chapter 5

Concluding remarks

In the previous chapters, we first describe the motivation behind our work – solving global optimization problems reliably with rigorous techniques. Then we review related techniques, such as local optimization and interval analysis, as well as a brief description of available global optimization solvers. The next two chapters show our contributions and their results: First, a speculative optimization algorithm that uses interval analysis to "bet" in the value of the objective function; second, a combination of techniques that contribute to the speculation, improving the convergence of the upper bound of the objective function for unconstrained and constrained optimization problems.

In this chapter, we provide concluding remarks and observations about the methods presented in this thesis, and we list venues for future work on potential improvements on speculative optimization.

5.1 Conclusions

In this work we presented a speculative optimization algorithm that aims to reduce the amount of splitting in comparison with traditional branch-and-prune algorithms. We measured the performance of the algorithm by comparing the accuracy and convergence of both algorithms towards the global minimum.

For unconstrained optimization, the initial speculative algorithm (Algorithm 1) does not find a solution in reasonable time to all the problems that branch-and-prune (Algorithm 0) solves consistently within the same time frame. The results provided by Algorithm 1 on those benchmarks it solved before timing out are as accurate as the same results from

Algorithm 0, which means they enclose the solution using intervals. On average, for those problems that can be solved by both algorithms, Algorithm 1 completes its search faster than Algorithm 0.

Both algorithms time out without completing the search, reporting partial results in the majority of the constrained optimization benchmarks. Algorithm 0 finds closer solutions for more problems than Algorithm 1.

These results motivated the introduction of additional techniques for unconstrained and constrained problems. For unconstrained problems, Algorithm 2 incorporates local optimization and monotonicity analysis to reduce the size of the domain and improve the upper bound of the objective function. For constrained problems, Algorithm creates a linear relaxation of the problem using Karush-Kuhn-Tucker conditions, which are solved using interval Newton method to obtain a new objective function upper bound.

Algorithm 2 solved the great majority of the unconstrained benchmarks, enclosing the solution with accuracy under less computation time than Algorithm 0. Based on these results we conclude that in general the addition of monotonicity analysis and local optimization results in increased accuracy with reduced time.

For constrained optimization, Algorithm 2 showed little improvement in both time and accuracy over Algorithms 0 and 1 on a limited number of problems. Our conclusion is that the KKT linear problem solved with interval Newton method will improve time and accuracy in a limited number of problems.

5.2 Future work

Based on the results for Algorithm 2, the first idea is to expand the idea used to solve unconstrained optimization (analyzing the shape of the function), into constrained optimization, using techniques that incorporate both the objective function and the constraints, such as convexity analysis [28].

The second idea also focuses on the constrained optimization case. Penalty functions,

which transform a constrained optimization problem into an unconstrained one by using the constraints as penalties, are common in existing literature with good results [21, 25, 24].

The next potential step for the algorithm is to cut its dependency on an external solver, by implementing into the main algorithm the constraint consistency algorithms we use and that we currently call RealPaver to run (HC4, BC5, interval Newton) [14, 16]. This gives us more control over the consistency process and allows us to expand these algorithms to suit the needs of the problems we solve.

Finally, another future work area is to apply this algorithm to more specific types of problems, such as large-scale optimization problems. Large scale optimization problems have up to millions of variables and constraints, and methods that rely on domain splitting, such as branch-and-prune, have difficulty solving them.

Bibliography

[1] Krzysztof Apt. *Principles of constraint programming*. Cambridge University Press, 2003.

[2] Douglas N Arnold. "The patriot missile failure". In: *Retrieved October* 3 (2000), p. 2010.

[3] Mordecai Avriel. *Nonlinear programming: analysis and methods*. Courier Dover Publications, 2003.

[4] F Benhamou, D McAllester, and P Van Hentenryck. "CLP (Intervals) revisited". In: *Proc. of International Logic Programming Symposium, 1994* (1994).

[5] Frédéric Benhamou et al. "Revising hull and box consistency". In: *Int. Conf. on Logic Programming*. Citeseer. 1999.

[6] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2009.

[7] L. Carlozo. *Why college students stop short of a degree*. `http://johngress.com/2012/03/27/why-college-students-stop-short-of-a-degree`.

[8] Gilles Chabert. *IBEX, an Interval-Based EXplorer*. 2007.

[9] A. F. Garcia Contreras et al. "Interval Optimization to Predict Software Quality Assessment Decisions". In: *INFORMS Optimization Society Conference 2012*. Coral Gables, FL, 2012.

[10] Robert J Dakin. "A tree-search algorithm for mixed integer programming problems". In: *The Computer Journal* 8.3 (1965), pp. 250–255.

[11] L. Davis. *Genetic Algorithms and Simulated Annealing*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 1987.

[12] Patriot Missile Defense. "Software problem led to system failure at Dhahran, Saudi Arabia". In: *US GAO Reports, report no. GAO/IMTEC-92-26* (1992).

[13] John E Dennis Jr and Jorge J Moré. "Quasi-Newton methods, motivation and theory". In: *SIAM review* 19.1 (1977), pp. 46–89.

[14] Laurent Granvilliers. "RealPaver users manual". In: *IRIN, University of Nantes, 0.1 edition* (2002).

[15] Laurent Granvilliers. "Towards cooperative interval narrowing". In: *Frontiers of Combining Systems*. Springer, 2000, pp. 18–31.

[16] Laurent Granvilliers and Frédéric Benhamou. "Algorithm 852: Realpaver: an interval solver using constraint satisfaction techniques". In: *ACM Transactions on Mathematical Software (TOMS)* 32.1 (2006), pp. 138–156.

[17] I. Griva, S.G. Nash, and A. Sofer. *Linear and Nonlinear Optimization: Second Edition*. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2009. ISBN: 9780898717730.

[18] Eldon Hansen and G William Walster. *Global optimization using interval analysis: revised and expanded*. Vol. 264. CRC Press, 2003.

[19] ER Hansen and RI Greenberg. "An interval Newton method". In: *Applied Mathematics and Computation* 12.2 (1983), pp. 89–98.

[20] Frederick S Hillier. *Introduction to operations research*. Tata McGraw-Hill Education, 1995.

[21] Frank Hoffmeister and Joachim Sprave. "Problem-independent handling of constraints by use of metric penalty functions". In: (1996).

[22] Robert Hooke and To A Jeeves. ""Direct Search"Solution of Numerical and Statistical Problems". In: *Journal of the ACM (JACM)* 8.2 (1961), pp. 212–229.

[23] Thomas Huckle. "Collection of software bugs". In: *Institut für Informatik TU München: Munich, Germany. Available online: http://www5. in. tum. de/˜ huckle/bugse. html (accessed on 7 December 2012)* (2004).

[24] Waltraud Huyer and Arnold Neumaier. "A new exact penalty function". In: *SIAM Journal on Optimization* 13.4 (2003), pp. 1141–1158.

[25] Jeffrey A Joines and Christopher R Houck. "On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's". In: *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*. IEEE. 1994, pp. 579–584.

[26] Eric Jones, Travis Oliphant, and Pearu Peterson. "SciPy: Open source scientific tools for Python". In: *http://www. scipy. org/* (2001).

[27] Ralph Baker Kearfott. "Interval computations, rigour and non-rigour in deterministic continuous global optimization". In: *Optimization Methods & Software* 26.2 (2011), pp. 259–279.

[28] Ralph Baker Kearfott, Jessie Castille, and Gaurav Tyagi. "A general framework for convexity analysis in deterministic global optimization". In: *Journal of Global Optimization* 56.3 (2013), pp. 765–785.

[29] J. Kennedy and R. Eberhart. "Particle Swarm Optimization". In: *IEEE International Conference on Neural Networks*. Vol. 4. Perth, Australia, 1995, pp. 1942–1948.

[30] Slawomir Koziel, David Echeverría Ciaurri, and Leifur Leifsson. "Surrogate-based methods". In: *Computational Optimization, Methods and Algorithms*. Springer, 2011, pp. 33–59.

[31] Moritz Kuhn. "The Karush-Kuhn-Tucker Theorem". In: *CDSEM Uni Mannheim* (2006).

[32] M. Mathur et al. "Ant Colony Approach to Continuous Function Optimization". In: *Industrial & Engineering Chemistry Research* 39.10 (2000), pp. 3814–3822.

[33] Jean-Pierre Merlet. "ALIAS: an interval analysis based library for solving and analyzing system of equations". In: *SEA, June. Automation* (2000), pp. 1964–1969.

[34] Glenn E Miller. "Planning and scheduling the Hubble Space Telescope: Practical application of advanced techniques". In: *Artificial Intelligence, Robotics, and Automation for Space Symposium*. Vol. 1. 1994, pp. 339–343.

[35] Ramon E Moore. *Interval analysis*. Vol. 4. Prentice-Hall Englewood Cliffs, 1966.

[36] Ramon E Moore, R Baker Kearfott, and Michael J Cloud. *Introduction to interval analysis*. Siam, 2009.

[37] Ramon E Moore and RE Moore. *Methods and applications of interval analysis*. Vol. 2. SIAM, 1979.

[38] B Murtagh et al. *MINOS Solver Manual*. 2006.

[39] Arnold Neumaier. *Interval methods for systems of equations*. Vol. 37. Cambridge university press, 1990.

[40] Arnold Neumaier et al. "A comparison of complete global optimization solvers". In: *Mathematical programming* 103.2 (2005), pp. 335–356.

[41] Peter G Neumann. "The risks digest". In: *The Risks Digest* 4.41 ().

[42] Donald M Olsson and Lloyd S Nelson. "The Nelder-Mead simplex procedure for function minimization". In: *Technometrics* 17.1 (1975), pp. 45–51.

[43] D. Pham et al. "The bees algorithm-a novel tool for complex optimization problems". In: *Proceedings of 2nd International Virtual Conference on Intelligent Production Machines and Systems (IPROMS 2006)*. 2006, pp. 454–459.

[44] R. Poli, J. Kennedy, and T. Blackwell. "Particle swam optimization". In: *Swarm Intelligence* 1.1 (2007), pp. 33–57.

[45] Nestor V Queipo et al. "Surrogate-based analysis and optimization". In: *Progress in aerospace sciences* 41.1 (2005), pp. 1–28.

[46] Luis Miguel Rios and Nikolaos V Sahinidis. "Derivative-free optimization: A review of algorithms and comparison of software implementations". In: *Journal of Global Optimization* 56.3 (2013), pp. 1247–1293.

[47] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming*. Foundations of Artificial Intelligence. Elsevier Science, 2006. ISBN: 9780080463803.

[48] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, 2010. ISBN: 9780136042594.

[49] A. Ruszczynski. *Nonlinear Optimization*. Princeton University Press, 2011. ISBN: 9781400841059.

[50] N Sahinidis and M Tawarmalani. "BARON solver manual". In: *GAMS Development Corporation, Washington, DC, USA* (2009).

[51] Michel F Sanner et al. "Python: a programming language for software integration and development". In: *J Mol Graph Model* 17.1 (1999), pp. 57–61.

[52] O Shcherbina et al. "COCONUT benchmark". In: *Package downloadable from http://www. mat. univie. ac. at/˜ neum/glopt/coconut/Benchmark/Benchmark. html* (2009).

[53] Robert Skeel. "Roundoff error and the Patriot missile". In: *SIAM News* 25.4 (1992), p. 11.

[54] J. Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Applied Optimization. Springer, 2005. ISBN: 9780387243481.

[55] Peter Spellucci. "DONLP2 users guide". In: *TU Darmstadt. URL http://www. mathematik. tu-darmstadt. de/fbereiche/numerik/staff/spellucci/DONLP2* (2002).

[56] Robert Tarjan. "Depth-first search and linear graph algorithms". In: *SIAM journal on computing* 1.2 (1972), pp. 146–160.

[57] Pascal Van Hentenryck. *Numerica: a modeling language for global optimization*. MIT press, 1997.

[58] Pascal Van Hentenryck, David McAllester, and Deepak Kapur. "Solving polynomial systems using a branch and prune approach". In: *SIAM Journal on Numerical Analysis* 34.2 (1997), pp. 797–827.

[59] X. Wang and M. Ceberio. "Fuzzy Measure Extraction for Predicting At-Risk Students". In: *Proceedings of 2nd World Conference on Soft Computing*. Baku, Azerbaijan, 2012.

[60] SJ Wright and J Nocedal. *Numerical optimization*. Vol. 2. Springer New York, 1999.

Curriculum vita

Angel Fernando Garcia Contreras was born in Mexico City. The first son of Angel Fernando Garcia Barrientos and Maria Patricia Contreras Samberino, he graduated from the Tecnologico de Monterrey high school in Ciudad Juarez, Chihuahua, and afterwards entered the same institution to pursue a degree in Computer Systems Engineering in the fall of 2001. He graduated with honors in 2005, and proceeded to work for two years at the IT department of a twin plant, Critikon de Mexico, developing management information systems. In 2007 he joined Electronic Data Systems as a Software Analyst to work on the "Portal Microe" project. He joined Hewlett Packard Application Services when EDS was acquired by HP in 2008, where he worked on multiple software design for Mexican customs offices. In the spring of 2010, he entered Graduate School at The University of Texas at El Paso.

Permanent address: 212 Los Angeles St.
El Paso, Texas 79912-4927
Or
afgarciacontreras@miners.utep.edu

This thesis was typed by Angel Fernando Garcia Contreras.