Using the duality principle to improve lower bounds for the global minimum in nonconvex optimization

M.H. van Emden

Department of Computer Science,
University of Victoria, Victoria, Canada
vanemden@cs.uvic.ca
http://www.cs.uvic.ca/~vanemden/

1 Introduction

In 2001 Sergey Shary published "A Surprising Approach in Interval Global Optimization" [13], where he introduced

> . . . a new class of global optimization methods, called *graph subdivision methods*, that are based on the simultaneous adaptive subdivision of both the function's domain of definition and the range of values.

Shary considers the method "to turn out better than the traditional techniques from [11, 6, 8] in either the computational efficacy and the quality of the results it produces".

Although Shary presents it as an isolated phenomenon, it turns out that the graph subdivision method is intimately intertwined with a variety of methods in optimization. It is the purpose of this paper to present a unified framework in which the graph subdivision method can be appreciated. Foremost among these is the venerable principle of considering not only the primal formulation of the optimization problem, but also its dual. Hence the title.

2 Types of optimization problems considered here

We consider the minimization of f, a real-valued *objective function* of real-valued arguments $x_0, \ldots, x_{n-1}$, called *decision variables*:

$$\text{Minimize } f(x_0, \ldots, x_{n-1}) \text{ subject to } \langle x_0, \ldots, x_{n-1} \rangle \in S. \tag{1}$$

When $\langle x_0, \ldots, x_{n-1} \rangle \in S$, we say that $\langle x_0, \ldots, x_{n-1} \rangle$ is *feasible*. S is often described by means of *constraints*.

These can be equality constraints, as in:

$$h_0(x_0, \ldots, x_{n-1}) = 0, \ldots, h_{p-1}(x_0, \ldots, x_{n-1}) = 0 \tag{2}$$

They can also include inequality constraints, as in:

$$g_0(x_0, \ldots, x_{n-1}) \leq 0, \ldots, g_{q-1}(x_0, \ldots, x_{n-1}) \leq 0 \tag{3}$$

The standard formulation also includes the *set constraint*:

$$\langle x_0, \ldots, x_{n-1} \rangle \in \Omega \tag{4}$$

This includes possibly existing constraints that are not conveniently expressed as equality or inequality constraints; for example, that some or all of the decision variables have to be integer. Also, it is common for the decision vector to be constrained to a box, which is tedious to express in inequality constraints.

If $\{f(x_0, \ldots, x_{n-1}) \mid \langle x_0, \ldots, x_{n-1} \rangle \in S\}$ has a lower bound, then it has a greatest lower bound; call it m, the *global minimum*. Computationally, the optimization problem is to find an upper and a lower bound for the global minimum and to find a decision vector where the objective function attains a value close to m.

Optimization problems as described above may have several properties that facilitate solution by conventional methods: continuity of the objective function, absence of constraints, linearity of the objective function or the constraints, constraints in the form of equalities rather than inequalities, availability of partial derivatives of the objective function or constraints. Above all, the presence of only a single local minimum, or even convexity of the objective function, facilitates solution so much that large numbers of decision variables can be handled in the absence of constraints.

An important special case is *linear programming*, where the objective function is linear, where there are no equality constraints, and where the inequality constraints are also linear. Such optimization problems are efficiently solved by the Simplex algorithm.

We are primarily interested in methods that do not require any of these facilitating properties. In particular, we allow an unknown and possibly large number of local minima and we do not require the availability of derivatives. However, we do assume the following restrictive property: that the objective function and possibly existing constraints are given in the form of *expressions* that can be evaluated in interval arithmetic. This rules out minimization of objective functions that are the output of some "black box". For example, this would be the case if the objective function values are the result of a measurement.

Linear programming itself has an important special case, where variables are constrained to be integer. Then we speak of *integer linear programming* (if all variables are integer) or *mixed integer linear programming* (where some variables are integer and some need not be).

2.1 Intervals

As intervals play an important role in the optimization methods considered in this paper, we give a brief recapitulation of the most important interval concepts.

A *real interval* is a closed, connected set of reals. Accordingly, the empty set and the set of all reals are real intervals. The remaining real intervals can be classified as follows. Let a and b, with $a \leq b$, be reals. The bounded real intervals are $\{x \in \mathcal{R} \mid a \leq x \wedge x \leq b\}$. These are written as $[a, b]$. Real intervals that are only bounded on the left have the form $\{x \in \mathcal{R} \mid a \leq x\}$ and are written as $[a, +\infty]$. Real intervals that are only bounded on the right have the form $\{x \in \mathcal{R} \mid x \leq b\}$ and are written as $[-\infty, b]$. When we write $[-\infty, +\infty]$, we mean the set of all reals. The use of ∞ in these notations does not of course imply that $-\infty$ or $+\infty$ are claimed to be reals.

A *floating-point number* is any element of $F \cup \{-\infty, +\infty\}$, where F is a finite set of reals. A *floating-point interval* is a real interval, where the bounds, in so far as they exist, are floating-point numbers. A *canonical interval* is a non-empty floating-point interval that does not properly contain a floating-point interval.

For every non-empty interval X, $lb(X)$ and $rb(X)$ denote the left and right bound of X respectively. For an unbounded X, $lb(X)$ or $rb(X)$ is defined as $-\infty$ or $+\infty$. Thus, $X = [lb(X), rb(X)]$ is a convenient notation for all non-empty intervals, bounded or not.

A *box* is a cartesian product of intervals.

3 Determining upper and lower bounds

If the partial derivatives of the objective function are available and if constraints are absent, then there are conventional methods that efficiently find a *local minimum*. One then finds as many of these as possible and assumes that the least among the ones that are found is the global minimum. This is not in general the case, so we only have an *upper bound* for the global minimum.

For an upper bound of the global minimum one needs to find a feasible decision vector or one needs to prove the existence of one in the region of interest. This of course trivial in the absence of constraints; constraints can be such that it is difficult to obtain an upper bound. For example, constraints can take the form of nonlinear inequalities. Solving these is no less hard than the optimization problem. In fact, then the optimization problem might as well be stated as the finding of the best solution to a set of nonlinear inequalities, where "best" is defined by an objective function without special features that facilitate solution.

In the absence of constraints, finding an upper bound for the global minimum presents no difficulty. However, even in this simple case, conventional methods do not in general have a method for obtaining a lower bound. The approach of finding many local minima and stop with the hope of having found a low enough one breaks down in many problems. Some problems have astronomical numbers of local minima. In others the concept of local minimum even breaks down, as it does in integer linear programming.

Some special cases help in finding lower bounds. For example, if the objective function satisfies a Lipschitz condition; see [10]. In integer linear programming one can relax the problem by dropping the integer constraint. The remaining linear programming problem can be solved by the Simplex algorithm. The resulting minimum is then a lower bound for the global minimum. Both of these methods only yield useful lower bounds when the decision vector is confined to a small region. Hence it is necessary to subdivide the feasible region into many subregions. These regions are obtained by splitting on one of the n decision variables. Hence the number of these subregions is exponential in n.

It is of crucial importance to improve the upper and lower bounds as much as possible to postpone the inevitable combinatorial explosion as long as possible. In this paper we investigate Shary's proposal for improving the lower bounds.

4 Constraint Programming

The equality constraints (2) or inequality constraints (3) are difficult to solve if they are not linear. This difficulty has motivated interest in constraint programming. For example, in [7] it is shown that mixtures of nonlinear equalities and inequalities can be solved by interval constraints. This method has the property of bounding all solutions and, usually, isolating them. If the required derivatives are available then the necessary condition that they be zero at local minima can be added as constraints to the constraint satisfaction problem. By performing a branch-and-bound search over the typically many solutions of such a constraint satisfaction problem, these authors obtain impressive results in nonlinearly constrained nonconvex global optimization.

In this paper we address the situation where the objective function does not have the required derivatives or where these are not used for some other reason. In this case it is also advantageous for the finding of lower bounds of the objective function to cast the optimization problem stated in (1) in the form of a constraint satisfaction problem. This we treat in section 6. First, we start by briefly reviewing the main features of constraint programming.

Suppose we have information about variables in the form of constraints on the values these variables can assume. At any stage in the solution of the constraint satisfaction problem there is associated with each variable a *domain*, which is a set of values that the variable can have. In constraint programming one uses various methods to reduce the domains of the variables on the basis of the given constraints. After sufficient reduction, the domains are small enough to be a solution or to be used in lieu of a solution.

At first sight the equality and inequality constraints in (2) and (3) look like a constraint satisfaction problem. This is theoretically correct, as they completely determine the set of feasible vectors. However, they contain possibly deeply nested numerical expressions with possibly many variables. Such expressions are not suited for direct use in constraint programming.

The domains can only be reduced by removing values that do not satisfy one of the constraints, given that the other variables in that constraint are limited to their associated domains. The operation of removing such inconsistent values is the *domain reduction operation* (DRO) associated with that constraint. These operations are quickly computed, typically requiring at most a few dozen processor cycles. Such operations are only available for a small class of constraints that we call *primitive*. By introducing auxiliary variables, one can transform an equality or inequality with a nested expression such as (2) or (3) to a conjunction of primitive constraints.

Among the primitive constraints an important subclass consists of those that are the relational counterparts of the arithmetic operations, for example the ternary constraints $x + y = z$ (corresponding to addition and subtraction) and $x * y = z$ (corresponding to multiplication and division).

Thus it is no loss of generality to consider constraint satisfaction problems that are conjunctions of primitive constraints.

Definition 1. *A* constraint satisfaction problem (CSP) *consists of*

– *A non-empty set $\mathcal{V}$ of* variables.

– *A* domain vector *that associates each variable with a domain. Each domain is a set of which the elements can serve as value for the associated variable.*
– *A* set of constraints, *which are atomic formulas containing no variables other than those in* $\mathcal{V}$.

A *solution* is a choice of a domain element for each variable that makes all constraints true.

The constraint programming paradigm is very general. It applies to domains as different as booleans, integers, ground terms of logic, finite symbolic values, and reals. In this paper we consider *interval CSPs*, which are CSPs where there is only one type and it is equal to the set $\mathcal{R}$ of real numbers. In such CSPs domains are restricted to floating-point intervals. We regard the integers as a subset of the real numbers. This corresponds to the possibility of placing a constraint on a real variable that it be integer. In this way, this framework also includes mixed-integer problems.

Arc consistency For each relation symbol that occurs in a CSP, there is a DRO. A domain vector of a CSP is *arc consistent* if none of the DROs of a constraint in the CSP causes an interval in the domain vector to become smaller. The arc consistent state is reached by applying the constraints in a "fair" order, that is, in such a way that every one of them is applied an infinite number of times.

In practice, we are restricted to domains that are representable in a computer. As there are only a finite number of these, there is a finite initial subsequence of every fair sequence that ends in an arc consistent domain vector. By *propagation* we mean an algorithm that applies DROs a finite number of times in such a way that the arc consistent domain vector is reached.

We say that propagation "terminates in failure" when the resulting domain vector has an empty interval. This special case is of interest as it proves that the original domain vector contains no solution. The nonfailure outcome does not prove the existence of any solution.

Box consistency When subdividing the space of decision vectors to obtain a close lower bound for the global minimum, it often pays to spend additional effort to improve the intervals that can be obtained by propagation. This is possible with a relaxation algorithm that can further reduce an arc-consistent domain vector to a box-consistent domain vector [1, 7].

Box consistency is based on an operation on one interval that is called "box narrowing" [7]. By repeating the application of box narrowing on sufficiently many intervals sufficiently many times, no further interval reductions result. Then the domain vector has reached the *box-consistent* state.

Functional box narrowing We consider two types of box narrowing: functional and relational. We describe them for constraint satisfaction problems that derive from constraints such as those in (3), though they are not limited in this way.

Let g be a generic member of $\{g_0, \ldots, g_{q-1}\}$. Let E be one of the expressions that compute the function g. We write $E(a_0, \ldots, a_{n-1})$ for the expression resulting from substituting $a_0, \ldots, a_{n-1}$ for $x_0, \ldots, x_{n-1}$ in E. Here $a_0, \ldots, a_{n-1}$ can be reals or they can be intervals.

Let *eval* denote the result of evaluating the expression to which it is applied. If reals have been substituted for the variables, then the evaluation is in real arithmetic. For example, for reals $a_0, \ldots, a_{n-1}$ we have $\mathit{eval}(E(a_0, \ldots, a_{n-1})) = g(a_0, \ldots, a_{n-1})$.

One can also substitute intervals for the variables in E. In that case, the evaluation is in interval arithmetic. Suppose that $X_0, \ldots, X_{n-1}$ are intervals. Then we have

$$\mathit{eval}(E(X_0, \ldots, X_{n-1})) \supset \{g(x_0, \ldots, x_{n-1}) \mid x_0 \in X_0, \ldots, x_{n-1} \in X_{n-1}\}.$$

In [7], box consistency is computed by a relaxation algorithm implemented in interval arithmetic. The algorithm takes as input certain intervals $X_0, \ldots, X_{n-1}$ for the variables $x_0, \ldots, x_{n-1}$. We call the algorithm in [7] a *relaxation algorithm* because it improves the intervals for the variables one at a time while keeping the intervals for all the other variables fixed. This is similar to the relaxation algorithms of numerical analysis.

For simplicity of notation, but without loss of generality, let us assume that the interval for x_0 is to be improved on the basis of the fixed intervals $X_1, \ldots, X_{n-1}$ for the variables $x_1, \ldots, x_{n-1}$. This is done by means of the function $g_{X_1,\ldots,X_{n-1}}$ that maps an interval to an interval and is defined so that for all intervals X

$$g_{X_1,\ldots,X_{n-1}}(X) = eval(E(X, X_1, \ldots, X_{n-1})).$$

To improve the interval $X_0 = [\mathit{lb}(X_0), \mathit{rb}(X_0)]$ for x_0, suppose that for some a such that $a \in \mathit{rb}(X_0)$ we have that

$$\mathit{lb}(g_{X_1,\ldots,X_{n-1}}([a, \mathit{rb}(X_0)])) > 0. \tag{5}$$

In that case the interval for x_0 can be improved from X_0 to $[\mathit{lb}(X_0), a]$ if $\mathit{lb}(X_0) \leq a$. If, on the other hand, $\mathit{lb}(X_0) > a$, then it has been shown that no solution of $g(x_0, \ldots, x_{n-1}) \leq 0$ exists for $x_0 \in X_0, \ldots, x_{n-1} \in X_{n-1}$.

```
real function BRB1(a, b) {
    if (lb(g_{X1,...,Xn-1}([a, b])) > 0) return a;
    if ([a, b] is canonical) return b;
    //[a, b] not canonical, so has midpoint
    m := midpoint of [a, b]; // a < m < b
    m' := BRB1(m, b);
    if (m' > m) return m';
    // rightbound was improved all the way down to m
    // so maybe m can be improved some more
    return BRB1(a, m);
}
```

Fig. 1. A definition of function to improve the right bound of $[a, b]$ by "squashing" or "box-narrowing", as they are respectively called in [3] and [7]. BRB stands for "better right bound". If applied to both bounds, then the result is functional box consistency. If $\mathrm{BRB1}(a, b)$ returns an interval with a as left bound, then there is no solution in $[a, b]$.

Figure 1 shows a bisection algorithm that can be used to find the least floating-point a for which (5) holds, for fixed intervals $X_1, \ldots, X_{n-1}$. A similar bisection is used to improve the lower bound of X_0 using g and the fixed intervals $X_1, \ldots, X_{n-1}$. In general, repeating this process with the other arguments and with the other functions among $\{g_1, \ldots, g_{q-1}\}$ causes reductions of $X_1, \ldots, X_{n-1}$ and further reductions of X_0.

If the interval becomes empty, it is shown that no solution exists within the original domain vector $X_0, \ldots, X_{n-1}$.

Relational box narrowing In (5) the criterion that decides whether a is a possible and better upper bound for the interval for x_0 depends on interval arithmetic for evaluating the left-hand side. This idea was used in the `absolve` predicate of the interval constraint programming language BNR Prolog [2]. In [15] it is compared with functional box consistency.

Consider the interval CSP containing as constraints

$$\begin{aligned} g(x_0, \ldots, x_{n-1}) &\leq 0 \\ x_0 &> a \\ x_0 \in X_0, x_1 \in X_1, \ldots, x_{n-1} &\in X_{n-1} \end{aligned} \tag{6}$$

If propagation on this CSP fails, we have a as improved upper bound. Relational box narrowing is a bisection algorithm for finding the best such a.

The interval CSP (6) can be seen as adding the constraint $x_0 > a$ to the following interval CSP denoted by $\mathcal{C}$.

$$\begin{aligned} g(x_0, \ldots, x_{n-1}) &\leq 0 \\ x_0 \in X_0, x_1 \in X_1, \ldots, x_{n-1} &\in X_{n-1} \end{aligned}$$

The corresponding bisection algorithm is shown in Figure 2.

```
real function BRB2(a, b) {
    apply propagation on C with x_0 > a;
    if (result of propagation is failure) return a;
    if ([a, b] is canonical) return b;
    m := midpoint of [a, b];
    m' := BRB2(m, b);
    if (m' > m) return m';
    return BRB2(a, m);
}
```

Fig. 2. A definition of a function to improve the right bound of $[a, b]$ using relational box consistency. If $\mathrm{BRB2}(a, b)$ returns an interval with a as left bound, then there is no solution in $[a, b]$.

In both functional and relational box consistency it is true that, if an interval becomes empty, the algorithm terminates with failure. As with propagation, the failure

outcome is a proof that no solution is contained in the original domain vector. Nonfailure does not imply the existence of any solution.

Partial box consistency The system (3) is representative of what might be solved by box consistency. In general, the method results in smaller intervals than propagation. The required effort is greater: each box narrowing uses one of the functions $g_0, \ldots, g_{q-1}$ to reduce the interval for one of the variables. To achieve box consistency, one has to iterate box-narrowing over the functions as well as over the variables. And box-narrowing itself is a bisection iteration.

There are situations where some of the variables or functions are more important than the others. In that case one can achieve *partial box consistency* by iterating the box narrowing operation only over those functions or variables.

5 The Duality Principle in optimization

We have seen a fundamental asymmetry in constraint programming. This asymmetry is the same independently of whether propagation, functional box consistency or relational box consistency is used: *failure proves absence of solutions; nonfailure does not prove the existence of any solution.* This has a consequence for what is the best way to solve an optimization problem by means of constraint programming.

In optimization it is part of standard methodology to consider not only the direct form (1) of the optimization problem, but to use the Duality Principle to state both the primal and dual formulation. There are many examples where the dual form leads to a better algorithm.

Given an objective function f and a set S of feasible vectors x, a typical statement of the Duality Principle is the equivalence of the following problems:

Primal Minimize $f(x)$ subject to $x \in S$.
Dual Maximize z such that $x \in S$ implies $f(x) > z$.

As the primal problem is to find a vector of n components, a search algorithm is likely to be exponential in n. The dual problem is to find a single real z. Of course it may be exponential to determine whether for any particular $z = z'$ it is the case that $x \in S$ implies $f(x) > z'$. It turns out that this can often be determined by failure of an arc-consistency or of a box-narrowing algorithm, which in practice appears to have a complexity that only increases moderately with n. It is therefore promising to approach optimization via the dual problem.

To be able to use constraint propagation or relational box consistency, we need to translate the optimization problem to a constraint satisfaction problem, which is a formula of logic. Although it is only the dual problem that appears promising, we might as well translate both to logic:

Primal Find the least y such that $\exists x.[f(x) = y \wedge x \in S]$.
Dual Find the greatest z such that $\forall x.[x \in S \rightarrow f(x) > z]$.

Merely re-expressing the primal problem in logic gives it the same one-dimensional appearance as the dual one. It was therefore naive to rely on such appearances to prefer the dual formulation. However, we still have to translate to a CSP. It turns out that the dual problem translates to a CSP on which arc consistency or relational box consistency can be used. This does not appear to be the case for the primal problem. This is not surprising, because of the difference in quantifiers between the problems[1].

By applying well-known equivalences to the dual problem, we obtain:

Primal

$$\text{Find the least } y \text{ such that } \exists x.[f(x) = y \wedge x \in S].$$

Dual

$$\text{Find the greatest } z \text{ such that } \neg\exists x.[f(x) \leq z \wedge x \in S]. \tag{7}$$

From the point of view of constraint programming there is a crucial difference between the two problems. According to the asymmetry noted in section 4, constraint propagation with real-valued variables can never by itself prove the existence of a solution. It can, however, prove that no solution exists. For that reason, the dual formulation is better suited to constraint programming. We now address the question: how does one find the greatest z specified in (7)?

6 Optimization expressed in constraint programming

The reason for preferring the dual formulation is that the negation can be inferred from the failure outcome of an arc-consistency or box-consistency algorithm. The existential quantification is implied in the CSP. Therefore the CSP to be manipulated for global optimization should be equivalent to what remains after removing the negation and the existential quantification: $f(x) \leq z \wedge x \in S$. It is encouraging that this is already a conjunction. We need to ensure that the conjuncts themselves also translate to conjunctions.

Let us begin with the conjunct $f(x) \leq z$. Here f denotes the mathematical concept of function, however defined. To express this as a CSP, we need a definition of f in the form of an arithmetic expression E. However, E is not an atomic formula. To complete the translation of $f(x) \leq z$ to a conjunct in a CSP, we add the variable f to the variables $x_0, \ldots, x_{n-1}$ that occur in E and in the equality and inequality constraints. This additional variable represents the value of the objective function. Thus we get

$$f = E \wedge f \leq z \wedge x \in S \tag{8}$$

as the formula specifying the CSP.

Let us now translate the conjunct $x \in S$ to a CSP. We assume that the set S is specified as the set of solutions to a conjunction of equality constraints (2) or inequality constraints (3). This conjunction then replaces the $x \in S$ in (8).

[1] The concept of duality also occurs in logic, where $\exists$ and $\forall$ are mutually dual.

We now have a CSP in the variables $x_0, \ldots, x_{n-1}, f$. From the point of view of constraint programming all these variables have the same status. In optimization, the conventional view sees constraints as limiting the values that can be assumed by $x_0, \ldots, x_{n-1}$. But when viewed as a CSP, the special status of f disappears. This is the insight that motivated the paper [13] by Shary. It may have been the contrast with the conventional view that prompted the "surprising" in his title.

Shary did not refer to constraint programming. He removed the special status of f by considering the "graph"

$$\{\langle x_0, \ldots, x_{n-1}, z\rangle \mid f(x_0, \ldots, x_{n-1}) = z\}$$

rather than the function f. He finds z as a lower bound for the global minimum by showing somehow that the equation $f(x_0, \ldots, x_{n-1}) = z$ has no solution. This depends of course on continuity of f. We use instead the inconsistency of $f(x_0, \ldots, x_{n-1}) \leq z$ because it is the formulation of the optimization problem itself, provided one uses the dual version. In this general formulation, continuity of f is not assumed. Shary did not elaborate on the crucial step of showing inconsistency of equations of the form $f(x_0, \ldots, x_{n-1}) = z$. He did refer to the possibility of using the interval constraint programming system Unicalc [12].

Using box narrowing The dual form (7) of the optimization problem suggests an iteration to determine the greatest z such that the CSP in (7) has no solution. But that is exactly the same mechanism used by box narrowing to improve the lower bound for f in the CSP $f = E \wedge x \in S$.

However, it must be realized that box narrowing finds z_1, the greatest z for which (7) fails. This is not z_0, the greatest z for which (7) has no solution. The good news is that $z_1 \leq z_0$. The bad news is that there can be quite a gap between the two. Typically, the smaller the domains for $x_0, \ldots, x_{n-1}$, the smaller the gap. Although the use of box-narrowing was introduced to decrease the need to subdivide the intervals for $x_0, \ldots, x_{n-1}$, this cannot be avoided altogether. Surprising and promising as the new technique is, it still has the humble status as one of the ways in which the venerable branch-and-bound method can be improved.

7 Branch-and-Bound

All methods that have gone beyond conventional continuous optimization in providing a lower bound for the global minimum use the branch-and-bound idea. These include [11, 6, 8] as well as Shary's graph-subdivision method and methods based on the use of a Lipschitz condition [10].

The many forms of branch-and-bound algorithms have in common that they depend on only a few parameters. By instantiating each of these, one obtains a branch-and-bound algorithm for a specific situation. The parameters are:

– *Bisection method* The feasible space of an optimization problem can be bisected unless it is "admissible"; see below.

- *Admissibility criterion* The feasible space consists of one decision vector or is small enough to be accepted in lieu of a solution.
- *Test method* An efficient algorithm that can have two outcomes of which one is *failure*. This outcome implies that no feasible decision vector exists. In case of nonfailure, there may be any number of solutions, including zero.
- *Lower-bound method* For every set of constraints there is a way to establish a number L such that there is no solution for which the objective function has a value less than L.
- *Upper-bound method* For every set of constraints there is a way to establish a number U such that there is no solution for which the objective function has a value greater than U. Such a bound is usually provided by the objective function value at a feasible point. It may not be possible to determine such a point. In that case the upper bound method yields $+\infty$.

As a result of repeated splitting, many optimization problems are generated, each with their own upper and lower bounds. If the lower bound in one problem exceeds the upper bound in another, then the former problem can be discarded from the list of problems under consideration. The success of branch-and-bound depends on making the lower bounds sharp enough to cause sufficiently many subproblems to be discarded.

Integer linear programming is an example of a branch-and-bound method. Lower bounds are obtained by dropping the integrality constraint and applying the Simplex algorithm to the remaining linear programming problem. If this yields an all-integer solution, the admissibility criterion is fulfilled. A feasible point is found, and this yields an upper bound. If the solution is not all integers, then a non-integer value a_j is found for a variable x_j and a split is effected by adding either the additional constraint $x_j \leq \lfloor a_j \rfloor$ or $x_j \geq \lceil a_j \rceil$.

We mention integer linear programming because it gave rise to the first branch-and-bound algorithm. The extensive experience over several decades has shown that the quality of lower bounds is crucial. Much effort has gone into devising cutting-plane techniques that effect small improvements in the lower bound. The seemingly trivial difference in lower bound is often needed to reduce the search space sufficiently to make the problem solvable in practice.

In nonlinear optimization problems, interval arithmetic has provided a way of providing lower bounds. This was done by the Moore-Skelboe algorithm [14, 9] and many subsequent elaborations [11, 6, 8]. Here the parameters for branch-and-bound are as follows. Bisection is the bisection of an interval that is not yet narrow enough. Admissibility can either be the fact that the difference between the upper and lower bounds is small enough or that all the intervals for the decision vector are small enough. The upper bound is the upper bound of the interval for the objective function obtained at a feasible point. This is of course problematic in constrained optimization problems. The lower bound parameter is the lower bound of the interval obtained by interval arithmetic evaluation of an expression for the objective function. The weakness of these lower bounds has motivated subsequent work on the use of constraint programming, including the present paper.

Unicalc [12] has been credited by Shary [13] as using constraint programming for improving lower bounds. Chen and van Emden [5, 4] implemented a branch-and-bound

algorithm based on the formulation (8) in the interval constraint programming language BNR Prolog [2]. The routine `absolve` was used to obtain a lower bound. Comparison with the Moore-Skelboe algorithm confirmed the experience in integer linear programming that improved lower bounds are often worth the additional expense by drastic reductions in the search space.

8 Conclusions

While the crucial importance of lower bounds in branch-and-bound has long been known, it is surprising that standard constraint programming techniques such as box-narrowing have been neglected in the literature on nonlinear nonconvex global optimization.

9 Acknowledgements

We have profited from discussions with Krzysztof Apt on branch-and-bound algorithms. Luc Jaulin and the anonymous referees have provided helpful remarks on an earlier draft. We acknowledge generous support by the University of Victoria, the Natural Science and Engineering Research Council NSERC, the Centrum voor Wiskunde en Informatica CWI, and the Nederlandse Organisatie voor Wetenschappelijk Onderzoek NWO.

References

1. F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) revisited. In *Logic Programming: Proc. 1994 International Symposium*, pages 124–138, 1994.
2. BNR. BNR Prolog user guide and reference manual. Version 3.1 for Macintosh, 1988.
3. Pascal Brisset, Hani El Sakkout, Thom Fruhwirth, Carmen Gervet, Warwick Harvey, Micha Meier, Stefano Novello, Thierry Le Provost, Joachim Schimpf, Kish Shen, and Mark Wallace. Eclipse constraint library manual. 2002.
4. H.M. Chen. *Global Optimization Using Interval Constraints*. PhD thesis, University of Victoria, 1998.
5. H.M. Chen and M.H. van Emden. Global optimization in hypernarrowing. In *SIAM Annual Meeting*, 1997.
6. Eldon Hansen. *Global Optimization Using Interval Analysis*. Marcel Dekker, 1992.
7. Pascal Van Hentenryck, Laurent Michel, and Yves Deville. *Numerica: A Modeling Language for Global Optimization*. MIT Press, 1997.
8. R. Baker Kearfott. *Rigorous Global Search: Continuous Problems*. Kluwer Academic Publishers, 1996. Nonconvex Optimization and Its Applications.
9. R.E. Moore. On computing the range of values of a rational function of n variables over a bounded region. *Computing*, 16:1–15, 1976.
10. János Pintér. *Global Optimization in Action*. Kluwer, 1996.
11. H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization*. Ellis Horwood/John Wiley, 1988.
12. Alexander L. Semenov. Solving optimization problems with help of the Unicalc solver. In R. Baker Kearfott and Vladik Kreinovich, editors, *Application of Interval Computations*, pages 211–224. Kluwer Academic Publishers, 1996.

13. Sergey P. Shary. A surprising approach in interval global optimization. *Reliable Computing*, 7:497–505, 2001.
14. S. Skelboe. Computation of rational interval functions. *BIT*, 14:87–95, 1974.
15. M.H. van Emden. Computing functional and relational box consistency by structured propagation in atomic constraint systems. In *Proc. 6th Annual Workshop of the ERCIM Working Group on Constraints; downloadable from CoRR*, 2001.