Global Optimization based on Contractor Programming: an Overview of the *IBEX* library

Jordan Ninin

ENSTA-Bretagne, LabSTIC, IHSEV team, 2 rue Francois Verny, 29806 Brest, France,
jordan.ninin@ensta-bretagne.fr

Abstract. *IBEX* is a open-source C++ library for constraint processing over real numbers. It provides reliable algorithms for handling non-linear constraints. In particular, roundoff errors are also taken into account. It is based on interval arithmetic and affine arithmetic. The main feature of *IBEX* is its ability to build strategies declaratively through the contractor programming paradigm. It can also be used as a black-box solver or with an AMPL interface. Two emblematic problems that can be addressed are: (i) **System solving**: A guaranteed enclosure for each solution of a system of (nonlinear) equations is calculated; (ii) **Global optimization**: A global minimizer of some function under non-linear constraints is calculated with guaranteed and reliable bounds on the objective minimum.

1 Kernel of *IBEX*

Considering sets in place of single points is not a common point of view in the Mathematical Programming communities. Unlike classical optimization tools, *IBEX* library relies on set-membership approach [1]. These methods and algorithms do not consider single numerical values, or floating-point numbers, but manipulate sets. The interval arithmetic offers a solid theoretical basis to represent and to calculate with subsets of $\mathbb{R}^n$.

1.1 Interval Arithmetic

An interval is a closed connected subset of $\mathbb{R}$. A non-empty interval $[\boldsymbol{x}]$ can be represented by its endpoints: $[\boldsymbol{x}] = [\underline{x}, \overline{x}] = \{x : \underline{x} \leq x \leq \overline{x}\}$ where $\underline{x} \in \mathbb{R} \cup \{-\infty\}$, $\overline{x} \in \mathbb{R} \cup \{+\infty\}$ and $\underline{x} \leq \overline{x}$. The set of intervals is denoted by $\mathbb{IR}$.

In *IBEX*, three external implementations of the Interval Arithmetic can be linked: *Filib++* [11], *Profil-Bias* [9], *Gaol* [8]. To improve the portability and the compatibility of *IBEX*, a homemade interval arithmetic is available without using low-level functionality which can be dependent of the architecture of the CPU. All the arithmetics have been patched to comply with the new IEEE 1788-2015 Standard for Interval Arithmetic.

1.2 Affine Arithmetic

Affine arithmetic is a technique to compute lower and upper bounds of functions over an interval. It is based on the same principle as interval arithmetic excepted that the quantities are represented by an affine form, see [15].

As in interval arithmetic, the usual operations and functions are extended to deal with affine forms. For the non-affine operations and some transcendental functions, such as the square root, the logarithm, the inverse and the exponential, several algorithms exist depending on the use: if you focus on computation of the bound, or on performance, or on linear approximation, or on the reliability, etc. Indeed, seven different versions are available to satisfy all needs of the user.

1.3 Contractor Programming

The concept of *contractor* is directly inspired by the ubiquitous concept of filtering algorithm in constraint programming [6]. The strength of *IBEX* lies mainly in this concept. Every algorithm in *IBEX* is included as a *Contractor*.

Definition 1. *Let $\mathbb{X} \subseteq \mathbb{R}^n$ be a feasible region.*
The operator $\mathcal{C}_{\mathbb{X}} : \mathbb{IR}^n \to \mathbb{IR}^n$ is a contractor for $\mathbb{X}$ if:

$$\forall [\boldsymbol{x}] \in \mathbb{IR}^n, \begin{cases} \mathcal{C}_{\mathbb{X}}([\boldsymbol{x}]) \subseteq [\boldsymbol{x}], & \textit{(contraction)} \\ \mathcal{C}_{\mathbb{X}}([\boldsymbol{x}]) \cap \mathbb{X} \supseteq [\boldsymbol{x}] \cap \mathbb{X}. & \textit{(completeness)} \end{cases}$$

This definition means that: (i) Filtering gives a sub-domain of the input domain $[\boldsymbol{x}]$; (ii) the resulting sub-domain $\mathcal{C}_{\mathbb{X}}([\boldsymbol{x}])$ contains all the feasible points. No solution is *lost*. A contractor is defined by a feasible region $\mathbb{X}$, and its purpose is to eliminate a part of a domain which is not in $\mathbb{X}$.

All set operators can be extended to contractors. For example, the intersection of two contractors creates a contractor for the intersection of these two sets. In the same way, the hull of two contractors creates a contractor for the disjunction of these constraints.

Definition 2. *Let $\mathbb{X}$ and $\mathbb{Y} \subseteq \mathbb{R}^n$ be two feasible regions.*

$$\begin{aligned} \texttt{CtcCompo:} \quad & (\mathcal{C}_{\mathbb{X}} \cap \mathcal{C}_{\mathbb{Y}})([\boldsymbol{x}]) = \mathcal{C}_{\mathbb{X}}([\boldsymbol{x}]) \cap \mathcal{C}_{\mathbb{Y}}([\boldsymbol{x}]) \\ \texttt{CtcUnion:} \quad & (\mathcal{C}_{\mathbb{X}} \cup \mathcal{C}_{\mathbb{Y}})([\boldsymbol{x}]) = \mathcal{C}_{\mathbb{X}}([\boldsymbol{x}]) \cup \mathcal{C}_{\mathbb{Y}}([\boldsymbol{x}]) \\ \texttt{CtcFixPoint:} \quad & \mathcal{C}^{\infty} = \mathcal{C} \circ \mathcal{C} \circ \mathcal{C} \circ \dots \end{aligned}$$

Using these properties, interacting, combining and merging heterogeneous techniques becomes simple.

2 Lists of Contractors

In this section, a small part of all contractors available in *IBEX* is described:

`CtcFwdBwd` and `CtcHC4`: the *atomic* contractors
Forward-backward is a classical algorithm in constraint programming for contracting quickly with respect to one equality or inequality constraint. See, e.g.,

[3], [7]. However, the more occurrences of variables in the expression of the (in)equality, the less accurate the contraction. Hence, this contractor is often used as an atomic contractor embedded in an higher-level operator like Propagation or Shaving.

HC4 is another classical algorithm of constraint programming. It allows to contract with respect to a system of constraints. The basic idea is to calculate the fix point of a set of n contractors $\mathcal{C}_1, \ldots, \mathcal{C}_n$, i.e. $(\mathcal{C} \circ \cdots \circ \mathcal{C})^\infty$, without calling a contractor when it is unnecessary.

`Ctc3BCid` and `CtcAcid`: the shaving contractors

`Ctc3BCID` is a shaving operator. It is an implementation of the 3BCID algorithm defined in [16]. The shaving operator applies a contractor $\mathcal{C}$ on sub-parts (*slices*) of the input box. If a slice is entirely eliminated by $\mathcal{C}$, the input box can be contracted by removing the slice from the box. This operator can be viewed as a generalization of the SAC algorithm in discrete domains [4]. The concept with continuous constraint was first introduced in [10] with the "3B" algorithm. In [10], the sub-contractor $\mathcal{C}$ was *CtcHC4*. In *IBEX*, the idea was extended and `Ctc3BCID` can be combined with every contractor.

`CtcAcid` is an adaptive version of the 3BCID contractor. The *handled* number of variables for which a shaving will be performed is adaptively tuned. The ACID algorithm alternates: (i) small tuning phases (during e.g 50 nodes) where the shaving is called on a number of variables double of the last tuned value (all variables in the first tuning phase); statistics are computed in order to determine an average number of interesting calls, The number of variables to be handled in the next running phase is computed at the end of the tuning phase; (ii) and large running phases (during e.g. 950 nodes) where 3BCID is called with the number of variables determined during the last tuning phase.

`CtcPolytopeHull`: Contractors based on Linear relaxation

Considering a system of linear inequalities, `CtcPolytopeHull` gives the possibility to contract a box to the hull of the polytope (the set of feasible points). This contractor calls a linear solver linked with *IBEX* (CPLEX, Soplex or CLP) to calculate for each variable x_i, the following bounds: $\min_{Ax \leq b \wedge x \in [\boldsymbol{x}]} x_i$ and $\max_{Ax \leq b \wedge x \in [\boldsymbol{x}]} x_i$, where $[\boldsymbol{x}]$ is the box to be contracted.

If some constraints are nonlinear, Linearization procedures can automatically linearize the non-linear constraints. There exists some built-in linearization techniques in *IBEX* :
(i) `LinearRelaxXTaylor`: a corner-based Taylor relaxation [2];
(ii) `LinearRelaxAffine2`: a relaxation based on affine arithmetic [14];
(iii) `LinearRelaxCombo`: a combination of the two previous techniques (the polytope is basically the intersection of the polytopes calculated by each technique).

`CtcQInter`: the q-relaxed intersection

If a set of constraints is based on physical data, it is not uncommon that some of this data is wrong. In this situation, the q-relaxed intersection of contractors can be applied to this problem.

The q-relaxed intersection of m subsets $\mathbb{X}_1, \ldots, \mathbb{X}_m$ of $\mathbb{R}^n$ is the set of all $x \in \mathbb{R}^n$ which belong to at least $(m-q)$ $\mathbb{X}_i$. We denote it by $\mathbb{X}^{\{q\}} = \bigcap^{\{q\}} \mathbb{X}_i$.

Since the q-relaxed intersection is a set operator, we have extended this notion to contractors: $\left(\bigcap^{\{q\}} \mathcal{C}_{\mathbb{X}_i}\right)([\boldsymbol{x}]) = \bigcap^{\{q\}} (\mathcal{C}_{\mathbb{X}_i}([\boldsymbol{x}]))$. This contractor allows modeling the possibility of invalid constraints: it can also be used for robust optimization.

In [5], Carbonnel et al. found an algorithm with a complexity $\theta(nm^2)$ to compute a box which contains the q-relaxed intersection of m boxes of $\mathbb{R}^n$.

`CtcExist` and `CtcForAll`: the contractors with quantifiers
Another possibility is to project a subset of $\mathbb{R}^n$ over one or more dimensions. For example, if a constraint needs to be satisfied for all values of a parameter in a given set, such as $\{x \in \mathbb{R}^n : \forall t \in \mathbb{X} \subseteq \mathbb{R}^m, g(x,t) \leq 0\}$, few solvers are available to deal with it. Another example is when a constraint needs to be satisfied for at least one value of the parameter, such as $\{x \in \mathbb{R}^n : \exists t \in \mathbb{X} \subseteq \mathbb{R}^m, g(x,t) \leq 0\}$.

Two operators are defined as contractors. The first one is `CtcForAll` and the second one is `CtcExist`. `CtcForAll` contracts each part of $[\boldsymbol{x}]$ which is contracted by $\mathcal{C}([\boldsymbol{x}] \times \{y\})$ for any $y \in \mathbb{Y}$. Indeed, each part $[\boldsymbol{a}]$ of $[\boldsymbol{x}]$, such as $\exists y \in \mathbb{Y},\ ([\boldsymbol{a}], y) \notin \mathbb{Z}$, can be removed. Thus, each part $[\boldsymbol{b}]$ of $[\boldsymbol{x}]$, such as $\forall y \in \mathbb{Y},\ ([\boldsymbol{b}], y) \in \mathbb{Z}$, is kept. A similar algorithm is used in `CtcExist`.

3 Optimization Strategies

To find the global optimum of a problem in a reliable way, *IBEX* included several global optimization strategies. The principle of these algorithms is based on a branch-and-bound technique [12]. At each iteration, the domain under study is bisected to improve the computation of bounds. Boxes are eliminated if and only if it is certified that no point in the box can produce a better solution than the current best one, or that at least one constraint cannot be satisfied by any point in such a box. To accelerate convergence, contractors are used at each iteration to prune the width of boxes.

The default optimization strategy is based on a mathematical model of the optimization problem that needs to be solved. This model can be constructed directly using the symbolic kernel of *IBEX* or using the AMPL interface. The default contractor inside is the following: $\texttt{CtcAcid} \cap (\texttt{CtcPolytopeHull} \cap \texttt{CtcHC4})^{\infty}$.

The performance of the default optimizer is comparable to the global optimizer *BARON*. However, our approach is completely reliable, and can deal with more general problems (with trigonometric function).

Moreover, a general pattern is also available [13]. This pattern only requires a contractor defined on the feasible set $\mathbb{X}$ of the problem and an other contractor on the unfeasible set $\overline{\mathbb{X}}$. Indeed, *IBEX* can address more complex real-life problems with disjunction constraint, quantifier, outliers, non-linearities, trigonometric functions, etc.

References

1. *IBEX*: a C++ numerical library based on interval arithmetic and constraint programming. http://www.ibex-lib.org.
2. I. Araya, G. Trombettoni, and B. Neveu. A contractor based on convex interval Taylor. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, pages 1–16. Springer, 2012.
3. F. Benhamou and L. Granvilliers. Continuous and interval constraints. *Handbook of Constraint Programming*, 2:571–603, 2006.
4. C. Bessiere and R. Debruyne. Theoretical analysis of singleton arc consistency. In *Proceedings of ECAI-04 workshop on Modeling and Solving Problems with Constraints*, 2004.
5. C. Carbonnel, G. Trombettoni, P. Vismara, and G. Chabert. Q-intersection Algorithms for Constraint-Based Robust Parameter Estimation. In *AAAI'14-Twenty-Eighth Conference on Artificial Intelligence*, pages 26–30, 2014.
6. G. Chabert and L. Jaulin. Contractor programming. *Artificial Intelligence*, 173(11):1079–1100, 2009.
7. H. Collavizza, F. Delobel, and M. Rueher. A Note on Partial Consistencies over Continuous Domains. In M. Maher and J.-F. Puget, editors, *Principles and Practice of Constraint Programming CP98*, number 1520 in Lecture Notes in Computer Science, pages 147–161. Springer Berlin Heidelberg, 1998.
8. F. Goualard. *Gaol: NOT Just Another Interval Library.* University of Nantes, France, 2005.
9. O. Knuppel. PROFIL/BIAS–A fast interval library. *Computing*, 53(3-4):277–287, 1994.
10. O. Lhomme. Consistency techniques for numeric CSPs. In *IJCAI*, volume 93, pages 232–238. Citeseer, 1993.
11. M. Nehmeier and J. Wolff v Gudenberg. FILIB++, Expression Templates and the Coming Interval Standard. *Reliable Computing*, 15(4):312–320, 2011.
12. J. Ninin. *Optimisation Globale basée sur l'Analyse d'Intervalles : Relaxation Affine et Limitation de la Mémoire.* PhD thesis, Institut National Polytechnique de Toulouse, Toulouse, 2010.
13. J. Ninin and G. Chabert. Global optimization based on contractor programming. In *XII GLOBAL OPTIMIZATION WORKSHOP*, pages 77–80, 2014.
14. J. Ninin, F. Messine, and P. Hansen. A reliable affine relaxation method for global optimization. *4OR*, 13(3):247–277, 2014.
15. J. Stolfi and L.H. De Figueiredo. *Self-validated numerical methods and applications.* Monograph for 21st Brazilian Mathematics Colloquium. IMPA/CNPq, Rio de Janeiro, Brazil, 1997.
16. G. Trombettoni and G. Chabert. Constructive interval disjunction. In *Principles and Practice of Constraint Programming–CP 2007*, pages 635–650. Springer, 2007.