Simple and Effective Sign Consistency Using Interval Arithmetic

Federico Bergenti and Stefania Monica

Dipartimento di Scienze Matematiche, Fisiche e Informatiche
Università degli Studi di Parma, 43124 Parma, Italy
{federico.bergenti,stefania.monica}@unipr.it

Abstract. Polynomial constraints over finite domains are expressed as equalities, inequalities, and disequalities of polynomials with integer coefficients whose variables take values from finite subsets of the integers. They are an interesting type of constraints that can be used to model many combinatorial problems over the integers. Sign consistency is a type of local consistency proposed specifically to support reasoning on polynomial constraints over finite domains. Sign consistency is parameterized in terms of a bounding function that is used to extract relevant information from a constraint when its variables are restricted to take values from a finite box. Known results from the literature on interval arithmetic can be readily used to propose a simple bounding function to support sign consistency. Preliminary experimental results show that the proposed bounding function is a promising candidate to effectively reason on polynomial constraints over finite domains.

Keywords: Sign consistency · Bounding functions · Polynomial constraints over finite domains · Constraint satisfaction problems.

1 Introduction

Polynomials are ubiquitous because they are the basis of important applications in virtually all fields of science and engineering. In addition, the influential literature on computer algebra witnesses the advanced level of understanding of polynomials from a computational point of view (e.g., [11]). Polynomial constraints, which are commonly regarded as constraints expressed in terms of equalities and inequalities of polynomials whose variables take values from subsets of the reals, are studied extensively in constraint satisfaction (e.g., [17]) and in constrained optimization (e.g., [18]). Polynomial constraints over finite domains [2] are polynomial constraints whose variables are restricted to take values from finite subsets of the integers. As such, they extend polynomial constraints with the addition of disequalities, which are not normally considered for real variables. Polynomial constraints over finite domains have been initially studied in the scope of constraint logic programming [2,4,5,6], but this paper shows that they can also be studied in the scope of constraint satisfaction problems, as anticipated in a previous preliminary work [3].

Sign consistency is defined and studied in an upcoming paper as a type of local consistency (e.g., [17]) specifically designed to take advantage of the peculiarities of polynomial constraints over finite domains. Sign consistency is parameterized in terms of a bounding function whose purpose is to extract relevant information from a given constraint when its variables are restricted to take values from a given box. Bounding functions are used to adapt sign consistency to the characteristics of studied problems, and to compromise its strength with the computational cost needed to enforce it. The definition of sign consistency is briefly reported in Section 4, and interested readers can obtain from authors a preprint of the paper where sign consistency is comprehensively studied. Relevant characteristics of sign consistency, which include its intimate relationship with hyper-arc consistency (e.g., [1]), are discussed in the mentioned paper.

Polynomial constraints over finite domains are expressed as equalities, inequalities, and disequalities of polynomials whose variables take values from finite subsets of the integers. Therefore, the satisfiability of one of such constraints can be studied in terms of the study of the sign of a polynomial function. Sign consistency uses this approach to reason on polynomial constraints over finite domains. In particular, sign consistency delegates the study of the sign of polynomial functions to a bounding function that is chosen to adapt it to the characteristics of the studied problem. The chosen bounding function is expected to provide effective lower and upper bounds of studied polynomial functions when their variables take values from given boxes.

Various bounding functions with diverse characteristics can be readily defined. A very simple bounding function can be defined in terms of the exhaustive enumeration of the elements of the given box, which is always finite because the variables of studied constraints take values from finite subsets of the integers. Such a bounding function is obviously impractical even for moderately complex problems. A less trivial bounding function can be defined using classic results on the Bernstein form of polynomials (e.g., [7] for a survey of recent results and a historical retrospective). The use of such a bounding function is expected to be applicable in practical situations only if advanced algorithms to compute the Bernstein form of polynomials are adopted (e.g., [8,10,13,14,20]). Finally, a third bounding function can be defined on the basis of a generalization for several variables [3] of known results for one [16] and two [9] real variables whose values are restricted to the closed interval $[0, 1]$. The computational cost associated with such a bounding function is expected to be lower than that of the other two mentioned alternatives because it evaluates the given polynomial function only at the corners of the considered box.

Even if mentioned bounding functions can be used to support sign consistency with no restrictions on the considered constraints or on the considered domains of variables, preliminary results suggest that their inherent computational cost is not compatible with practical problems. For this reason, this paper proposes a simple bounding function whose purpose is to make sign consistency a valuable tool to solve practical constraint satisfaction problems that include polynomial constraints over finite domains.

As discussed in Section 4, the bounding function proposed in this paper is based on classic results from the literature on interval arithmetic (e.g., [21]). The lower and upper bounds computed using such a bounding function are expected to be less stringent than those computed using one of the three bounding functions mentioned above, but its simplicity and low computational cost contribute to make it a viable option for practical problems. Preliminary experimental results on the use of the proposed bounding function to solve a few selected constraint satisfaction problems taken from a well-known library of problems are presented in Section 5. In detail, a prototype implementation of a specific algorithm to solve constraint satisfaction problems using the proposed bounding function was included in PolyFD, which is an experimental C++ library for constraint satisfaction. Then, specific programs that take advantage of PolyFD were used to solve the selected problems and to gather preliminary experimental results. Obtained experimental results are encouraging because they show that the time needed to solve studied problems using PolyFD is within the same order of magnitude of the time needed to solve the same problems using Gecode [19].

This paper is organized as follows. Section 2 provides a self-contained presentation of adopted notation, which includes the notation borrowed from interval arithmetic. Section 3 presents polynomial constraints over finite domains in the scope of constraint satisfaction problems. Section 4 introduces sign consistency and discusses the proposed bounding function. Section 5 shows preliminary experimental results on the use of the proposed bounding function to solve a few selected constraint satisfaction problems. Finally, Section 6 concludes the paper and provides a brief overview of future research directions.

2 Notation and Preliminaries

This section is intended to provide a self-contained report on adopted notation. The adopted notation is based on the ordinary notation used to study polynomial functions of several variables, and it includes the characteristic notation of interval arithmetic. Note that, together with the common notation used to study polynomial functions of real variables, a specific notation is also introduced to study polynomial functions with integer coefficients whose variables take values from finite subsets of the integers.

The set of natural numbers, which includes zero, is denoted as $\mathbb{N}$, while the set of positive natural numbers is denoted as $\mathbb{N}_+$. Given $n \in \mathbb{N}_+$, a multi-index $I \in \mathbb{N}^n$ is defined as an $n-$tuple of natural numbers

$$I = (i_1, i_2, \ldots, i_n) = (i_k)_{k=1}^n, \tag{1}$$

where the common notation of using an uppercase letter to denote a multi-index, and the same letter, but lowercase and with a subscript, for its components is used. The following notation is adopted to use a multi-index $I \in \mathbb{N}^n$ as an exponent for $\mathbf{t} \in \mathbb{R}^n$ with $\mathbf{t} = (t_1, t_2, \ldots, t_n) = (t_k)_{k=1}^n$

$$\mathbf{t}^I = \prod_{k=1}^{n} t_k^{i_k}, \tag{2}$$

where the common notation of using a boldface letter to denote an $n-$tuple of real numbers, and the same letter, but lightface and with a subscript, for its components is used. Note that the common understanding of $0^0 = 1$ is adopted consistently in this paper.

Given two multi-indices $I \in \mathbb{N}^n$ and $J \in \mathbb{N}^n$, they can be compared using the following partial order

$$I \leq J \iff i_1 \leq j_1 \wedge i_2 \leq j_2 \wedge \cdots \wedge i_n \leq j_n \tag{3}$$

and they can be added componentwise

$$I + J = (i_1 + j_1, i_2 + j_2, \ldots, i_n + j_n). \tag{4}$$

The following abbreviation is introduced for multi-indices $H \in \mathbb{N}^n$, $I \in \mathbb{N}^n$, and $J \in \mathbb{N}^n$ to express repeated sums

$$\sum_{H \leq I \leq J} (\cdot) = \sum_{i_1=h_1}^{j_1} \sum_{i_2=h_2}^{j_2} \cdots \sum_{i_n=h_n}^{j_n} (\cdot). \tag{5}$$

Note that H is omitted in (5), so that only $I \leq J$ remains, if it is the null multi-index $(0)_{k=1}^n$.

Using the multi-index notation, a polynomial function $p : \mathbb{R}^n \mapsto \mathbb{R}$ of $n \in \mathbb{N}_+$ real variables with real coefficients can be expressed as

$$p(\mathbf{x}) = \sum_{I \leq L} a_I \mathbf{x}^I, \tag{6}$$

where $\{a_I\}_{I \leq L} \subset \mathbb{R}$ is the set of the coefficients of p, and multi-index $L \in \mathbb{N}^n$ is the multi-degree of p. The vector space of polynomial functions of $n \in \mathbb{N}_+$ real variables, with real coefficients, and with multi-degree less than or equal to multi-index $L \in \mathbb{N}^n$ is

$$\Pi_L = \text{span}_{\mathbb{R}}\{P_I\}_{I \leq L} \qquad P_I : \mathbb{R}^n \mapsto \mathbb{R} \quad P_I(\mathbf{x}) = \mathbf{x}^I. \tag{7}$$

Similarly, a polynomial function $\tilde{p} : \mathbb{Z}^n \mapsto \mathbb{Z}$ of $n \in \mathbb{N}_+$ integer variables with integer coefficients can be expressed as

$$\tilde{p}(\mathbf{x}) = \sum_{I \leq L} \tilde{a}_I \mathbf{x}^I, \tag{8}$$

where $\{\tilde{a}_I\}_{I \leq L} \subset \mathbb{Z}$ is the set of the coefficients of $\tilde{p}$, and multi-index $L \in \mathbb{N}^n$ is the multi-degree of $\tilde{p}$. The vector space of polynomial functions of $n \in \mathbb{N}_+$ integer variables, with integer coefficients, and with multi-degree less than or equal to multi-index $L \in \mathbb{N}^n$ is

$$\tilde{\Pi}_L = \text{span}_{\mathbb{Z}}\{\tilde{P}_I\}_{I \leq L} \qquad \tilde{P}_I : \mathbb{Z}^n \mapsto \mathbb{Z} \quad \tilde{P}_I(\mathbf{x}) = \mathbf{x}^I. \tag{9}$$

Given that the major interest in this paper is on the study of the bounds of polynomial functions over boxes, the following notation for intervals and boxes is recalled. A closed (real) interval from $\underline{a} \in \mathbb{R}$ to $\overline{a} \in \mathbb{R}$ is denoted as

$$[\underline{a}, \overline{a}] = \{x \in \mathbb{R} : \underline{a} \leq x \leq \overline{a}\}, \tag{10}$$

and it equals the empty set if and only if $\underline{a} > \overline{a}$. A singleton (real) interval that contains only $a \in \mathbb{R}$ is denoted as $[a] = [a, a]$. Given $n \in \mathbb{N}_+$, a (real) box $B \subset \mathbb{R}^n$ from $\underline{\mathbf{b}} = (\underline{b}_k)_{k=1}^n \in \mathbb{R}^n$ to $\overline{\mathbf{b}} = (\overline{b}_k)_{k=1}^n \in \mathbb{R}^n$ is denoted as

$$B = [\underline{\mathbf{b}}, \overline{\mathbf{b}}] = [\underline{b}_1, \overline{b}_1] \times [\underline{b}_2, \overline{b}_2] \times \cdots \times [\underline{b}_n, \overline{b}_n], \tag{11}$$

and it equals the empty set if and only if $\underline{b}_i > \overline{b}_i$ for some $1 \leq i \leq n$. Note that the notation $B_i = [\underline{b}_i, \overline{b}_i] \subset \mathbb{R}$ with $1 \leq i \leq n$ is used to refer to the closed intervals that compose the nonempty box B. Also note that the notation $B_{i \to S}$ is used to refer to the box obtained by replacing the $i-$th closed interval that composes the nonempty box B with the nonempty closed interval $S \subset \mathbb{R}$.

The notation just recalled for intervals and boxes is often adapted to refer to integer intervals and integer boxes, as follows. An integer interval from $\underline{c} \in \mathbb{Z}$ to $\overline{c} \in \mathbb{Z}$ is denoted as

$$[\underline{c}..\overline{c}] = \{x \in \mathbb{Z} : \underline{c} \leq x \leq \overline{c}\}, \tag{12}$$

and it equals the empty set if and only if $\underline{c} > \overline{c}$. A singleton integer interval that contains only $c \in \mathbb{Z}$ is denoted as $[c] = [c..c]$. Given $n \in \mathbb{N}_+$, an integer box $D \subset \mathbb{Z}^n$ from $\underline{\mathbf{d}} = (\underline{d}_k)_{k=1}^n \in \mathbb{Z}^n$ to $\overline{\mathbf{d}} = (\overline{d}_k)_{k=1}^n \in \mathbb{Z}^n$ is denoted as

$$D = [\underline{\mathbf{d}}..\overline{\mathbf{d}}] = [\underline{d}_1..\overline{d}_1] \times [\underline{d}_2..\overline{d}_2] \times \cdots \times [\underline{d}_n..\overline{d}_n], \tag{13}$$

and it equals the empty set if and only if $\underline{d}_i > \overline{d}_i$ for some $1 \leq i \leq n$. Note that the notation $D_i = [\underline{d}_i..\overline{d}_i] \subset \mathbb{Z}$ with $1 \leq i \leq n$ is used to refer to the integer intervals that compose the nonempty box D. Also note that the notation $D_{i \to T}$ is used to refer to the box obtained by replacing the $i-$th integer interval that composes D with the nonempty integer interval $T \subset \mathbb{Z}$. Finally, the bounding box $\square A$ of a nonempty finite integer set $A \subset \mathbb{Z}^n$ is the inclusion-minimal integer box such that $A \subseteq \square A$.

Interval arithmetic (e.g., [21]) uses the introduced notation to express computations whose arguments and results are closed intervals. In particular, arithmetic operations on nonempty closed intervals are normally defined in interval arithmetic as follows. Given two nonempty closed intervals $A = [\underline{a}, \overline{a}] \subset \mathbb{R}$ and $C = [\underline{c}, \overline{c}] \subset \mathbb{R}$, they can be added

$$A + C = [\underline{a} + \underline{c}, \overline{a} + \overline{c}], \tag{14}$$

and they can be multiplied

$$A \cdot C = [\min\{\underline{a}\,\underline{c}, \underline{a}\,\overline{c}, \overline{a}\,\underline{c}, \overline{a}\,\overline{c}\}, \max\{\underline{a}\,\underline{c}, \underline{a}\,\overline{c}, \overline{a}\,\underline{c}, \overline{a}\,\overline{c}\}]. \tag{15}$$

In both cases, the result is a nonempty closed interval. Note that the product among intervals can be used to define the $m-$th power of a closed interval $A \subset \mathbb{R}$, where $m \in \mathbb{N}_+$. Also note that the convention $[0]^0 = [1]$ is adopted consistently in this paper. Given $n \in \mathbb{N}_+$, a nonempty box $B \subset \mathbb{R}^n$, and a multi-index $I \in \mathbb{N}^n$, the following notation that uses only products among intervals is adopted

$$B^I = \prod_{j=1}^{n} B_j^{i_j}. \tag{16}$$

Using the introduced notation, which is borrowed from interval arithmetic, a polynomial function $p : \mathbb{R}^n \mapsto \mathbb{R}$ of $n \in \mathbb{N}_+$ real variables with real coefficients can be extended to work on boxes. If polynomial p is expressed as in (6), then for any box $B \subset \mathbb{R}^n$

$$p(B) = \sum_{I \leq L} [a_I] B^I, \tag{17}$$

where the notation for singleton intervals is used to treat the coefficients of the polynomial function as closed intervals. Note that it is easy to prove that the result of the evaluation of $p(B)$ is a closed interval that includes the range of p over box B, which ensures that interval arithmetic can be used to perform validated computations (e.g., [21]).

Even if interval arithmetic is not normally used to express computations on integer intervals, the introduced notation can be easily generalized to integer intervals and integer boxes. Actually, straightforward generalizations to integer intervals of the arithmetic operations on closed intervals defined previously can be used to express the evaluation of a polynomial function $\tilde{p}$ of $n \in \mathbb{N}_+$ integer variables with integer coefficients when an integer box $D \subset \mathbb{Z}^n$ is used as argument. The result of such an evaluation is an integer interval that includes the range of $\tilde{p}$ over the integer box D.

3 Polynomial Constraints over Finite Domains

This section presents polynomial constraints over finite domains in the scope of constraint satisfaction problems. Note that polynomial constraints over finite domains have been already studied in the scope of constraint logic programming in previous papers [2,4,5,6]. The discussion in this section differs from the presentations in cited papers mostly because polynomial constraints over finite domains are presented in cited papers using a canonical form that is not adopted in this paper. Actually, in order to simplify the presentation, the canonical form adopted in cited papers expresses constraints using only inequalities, which ultimately requires to increase the multi-degrees of polynomials to treat disequalities. In this paper, a different form of constraints is used to avoid increasing the multi-degrees of polynomials. Note that the form of constraints adopted in this paper is still questionable because it requires to increase the number of constraints to treat equalities. In addition, note that the adopted form of constraints is not required to be unique, and therefore it is not canonical.

The following ordinary nomenclature on constraint satisfaction problems (e.g., [1]) is needed to formally introduce polynomial constraints over finite domains in Definition 2.

Definition 1 (Constraint, assignment, and satisfiability). *An $n-$ary constraint C whose $n \in \mathbb{N}_+$ variables take values from domains $(D_i)_{i=1}^n$ is a subset of the Cartesian product of the domains of variables*

$$C \subseteq \prod_{i=1}^{n} D_i. \tag{18}$$

An assignment $\mathbf{a}$ of the variables of C is an element of the Cartesian product of the domains of variables

$$\mathbf{a} \in \prod_{i=1}^{n} D_i. \tag{19}$$

An assignment of the variables of C satisfies C if and only if $\mathbf{a} \in C$. A constraint C is satisfiable if there exists at least one assignment of its variables that satisfies the constraint (i.e., $C \neq \emptyset$). On the contrary, a constraint C is unsatisfiable if no such an assignment exists (i.e., $C = \emptyset$).

The following definition captures the informal statement that describes polynomial constraints over finite domains as constraints expressed as equalities, inequalities, and disequalities of polynomials with integer coefficients whose variables take values from finite subsets of the integers.

Definition 2 (Polynomial constraint over finite domains). *An $n-$ary constraint C whose $n \in \mathbb{N}_+$ variables take values from domains $(D_i)_{i=1}^n$ is a polynomial constraint over finite domains if and only if all domains are finite subsets of $\mathbb{Z}$ and*

$$C = \{\mathbf{x} \in \prod_{i=1}^{n} D_i : p(\mathbf{x}) \geq 0\} \quad or \quad C = \{\mathbf{x} \in \prod_{i=1}^{n} D_i : p(\mathbf{x}) \neq 0\} \tag{20}$$

for a proper polynomial function $p \in \tilde{\Pi}_L$ of n integer variables, with integer coefficients, and with multi-degree less than or equal to multi-index $L \in \mathbb{N}^n$.

Observe that, with an abuse of notation that is normally accepted, the statements $p(\mathbf{x}) \geq 0$ and $p(\mathbf{x}) \neq 0$ can be used to refer to the corresponding constraints when the context makes the domains of variables evident.

Note that the two forms of polynomial constraints over finite domains identified in Definition 2 are not restrictive, and they can be used to refer to all equalities, inequalities, and disequalities among polynomials. Actually, the choice of such forms of constraints is justified by the following lemma, which is an adaptation of the corresponding lemma introduced in other papers (e.g., [2]) to support a canonical form for polynomial constraints over finite domains.

Lemma 1. *Given two polynomial functions $p \in \tilde{\Pi}_L$ and $q \in \tilde{\Pi}_L$ of $n \in \mathbb{N}_+$ integer variables, with integer coefficients, and with multi-degree less than or equal to multi-index $L \in N^n$, the following co-implications hold*

$$p(\mathbf{x}) \leq q(\mathbf{x}) \iff q(\mathbf{x}) - p(\mathbf{x}) \geq 0, \tag{21}$$
$$p(\mathbf{x}) < q(\mathbf{x}) \iff q(\mathbf{x}) - p(\mathbf{x}) - 1 \geq 0, \tag{22}$$
$$p(\mathbf{x}) > q(\mathbf{x}) \iff p(\mathbf{x}) - q(\mathbf{x}) - 1 \geq 0, \tag{23}$$
$$p(\mathbf{x}) \neq q(\mathbf{x}) \iff p(\mathbf{x}) - q(\mathbf{x}) \neq 0, \tag{24}$$
$$p(\mathbf{x}) = q(\mathbf{x}) \iff p(\mathbf{x}) - q(\mathbf{x}) \geq 0 \wedge q(\mathbf{x}) - p(\mathbf{x}) \geq 0. \tag{25}$$

Proof. The proof is trivially based on simple algebraic manipulations. □

Note that (22) and (23) are valid because p and q are polynomial functions of integer variables with integer coefficients, and they are not valid for ordinary polynomial functions of real variables. Also note that the application of (25) increases the number of constraints because it turns one equality constraint into two inequality constraints.

4 A Bounding Function Based on Interval Arithmetic

This section introduces sign consistency as a means to support reasoning on polynomial constraints over finite domains. In the first part of this section, sign consistency is defined as a form of local consistency parameterized in terms of a bounding function. In the second part of this section, a bounding function based on known results from the literature on interval arithmetic is proposed and briefly studied. The application of the proposed bounding function to reason on constraint satisfaction problems is discussed in Section 5, where preliminary experimental results are also presented and discussed.

4.1 Sign consistency

The following definition of bounding function is intended to support the successive definition of sign consistency.

Definition 3 (Bounding function). *A bounding function β is a computable function that, given a nonempty integer box $B \subset \mathbb{Z}^n$ and a polynomial function $p \in \tilde{\Pi}_L$ of $n \in \mathbb{N}_+$ integer variables, with integer coefficients, and with multi-degree less than or equal to multi-index $L \in \mathbb{N}^n$, computes $(\underline{p}, \overline{p}) \in \mathbb{R}^2$ such that the following conditions jointly hold*

$$\underline{p} \leq \min_{\mathbf{x} \in B} p(\mathbf{x}) \qquad \max_{\mathbf{x} \in B} p(\mathbf{x}) \leq \overline{p}. \tag{26}$$

The proposed definition of bounding function is sufficient to state the following parameterized definition of sign consistency.

Definition 4 (Sign consistency). *Given a bounding function β and a polynomial constraint over finite domains C whose $n \in \mathbb{N}_+$ variables take values from nonempty domains $(D_i)_{i=1}^n$, with $B = \square \prod_{i=1}^n D_i$, a value $v \in D_i$ with $1 \leq i \leq n$ is sign consistent for β with C if and only if*

1. *The constraint is $p(\mathbf{x}) \geq 0$, $\beta(p, B_{i \to [v..v]}) = (\underline{p}, \overline{p})$, and $\overline{p} \geq 0$; or*
2. *The constraint is $p(\mathbf{x}) \neq 0$, $\beta(p, B_{i \to [v..v]}) = (\underline{p}, \overline{p})$, and $\underline{p} \neq 0$ or $\overline{p} \neq 0$.*

A domain D_i with $1 \leq i \leq n$ is sign consistent for β with C if and only if all its values are sign consistent for β with C. Constraint C is sign consistent for β if and only if all its domains are sign consistent for β with C.

The recalled definition of sign consistency is parameterized in terms of a bounding function β, which is chosen to effectively reason on considered constraints. Three possible bounding functions are briefly mentioned in Section 1. A very simple bounding function can be computed by exhaustively enumerating the elements of the given box, which is always finite because the variables of studied constraints take values from finite subsets of the integers. Such a bounding function precisely computes the range of the studied polynomial function over the given box, but it is obviously impractical even for moderately complex problems. A less trivial bounding function can be computed using the Bernstein form of polynomials (e.g., [7] for a survey of recent results and a historical retrospective). The use of such a bounding function is expected to be feasible only if advanced algorithms (e.g., [8,10,13,14,20]) are adopted to compute the Bernstein form of polynomials. Finally, a third bounding function can be defined on the basis of a generalization for several variables [3] of known results for one [16] and two [9] real variables whose values are restricted to the closed interval $[0, 1]$. The computational cost of such a bounding function is expected to be lower than that of the other two alternatives because it computes the value of considered polynomial functions only at the corners of considered boxes. Unfortunately, the lower and upper bounds that it provides are typically less stringent than those provided by the other two alternatives, and therefore its effectiveness in the satisfaction of constraints is reduced.

4.2 The proposed bounding function

Even if the bounding functions briefly mentioned above do not impose restrictions on treated constraints or on the domains of variables, their application to solve practical constraint satisfaction problems is problematic for their inherent computational cost. Preliminary experiments on their use to reason on moderately complex constraints suggest that they are not effective in the solution of practical problems. For this reason, the following bounding function is proposed as a valid alternative to support the use of sign consistency to reason on practical constraint satisfaction problems. Note that the lower and upper bounds that it provides are typically less stringent than those provided by the three bounding functions mentioned above, but its simplicity and moderate computational cost make it a viable alternative for practical applications, as witnessed by the preliminary experimental results presented in Section 5.

Definition 5 (Bounding function β_I). *Given a nonempty integer box $B \subset \mathbb{Z}^n$ and a polynomial function $p \in \tilde{\Pi}_L$ of $n \in \mathbb{N}_+$ integer variables, with integer coefficients, and with multi-degree less than or equal to multi-degree $L \in \mathbb{N}^n$, the bounding function β_I is*

$$\beta_I(p, B) = (\underline{p}, \overline{p}), \tag{27}$$

where $p(B) = [\underline{p}..\overline{p}]$.

The following proposition is relevant because it is sufficient to prove that β_I complies with the definition of bounding functions.

Proposition 1. *Given a nonempty integer box $B \subset \mathbb{Z}^n$ and a polynomial function $p \in \tilde{\Pi}_L$ of $n \in \mathbb{N}_+$ integer variables, with integer coefficients, and with multi-degree less than or equal to multi-degree $L \in \mathbb{N}^n$,*

$$\underline{p} \leq \min_{\mathbf{x} \in B} p(\mathbf{x}) \qquad \max_{\mathbf{x} \in B} p(\mathbf{x}) \leq \overline{p}, \tag{28}$$

where $p(B) = [\underline{p}..\overline{p}]$.

Proof. The proof is trivially based on classic results from the literature on interval arithmetic (e.g. [21]). □

Note that Proposition 1 is not surprising because the computation of $p(B)$ using interval arithmetic is precisely intended to compute an integer interval that contains the range of the polynomial function p over the integer box B. Unfortunately, the computed interval often largely overestimates the range of the polynomial function p over the integer box B.

The following proposition is important because it quantifies the computational cost of the proposed bounding function. Note that the relevance of the proposed bounding function to reason on polynomial constraints over finite domains is motivated by its expected low computational cost.

Proposition 2. *Given a nonempty integer box $B \subset \mathbb{Z}^n$ and a polynomial function $p \in \tilde{\Pi}_L$ of $n \in \mathbb{N}_+$ integer variables, with integer coefficients, and with multi-degree less than or equal to multi-degree $L = (l_i)_{i=1}^n \in \mathbb{N}^n$, the bounding function β_I can be computed using $\mathcal{O}(l^n)$ arithmetic operations in the worst case, where $l = \max\{l_i\}_{i=1}^n$ is the greatest degree of a variable in p.*

Proof. The computation of β_I essentially equals the evaluation of a polynomial function, which can be performed using the multivariate Horner scheme (e.g., [14]). The proposition is proved because the multivariate Horner scheme requires $\mathcal{O}(l^n)$ arithmetic operations to evaluate the polynomial function p. □

Note that the lower and upper bounds computed by bounding function β_I can be improved using the following method to compute the $m-$th power of a closed interval when $m \in \mathbb{N}_+$ is even. Actually, it is easy to verify that the bounds computed using the following method are more stringent than the bounds computed in terms of successive multiplications. The method is a classic result

from the literature on interval arithmetic and it is rephrased here for the sake of completeness. In detail, given a closed interval $A = [\underline{a}, \overline{a}] \subset \mathbb{R}$, and given an even $m \in \mathbb{N}_+$

$$A^m = [\underline{a}, \overline{a}]^m = \begin{cases} [\underline{a}^m, \overline{a}^m] & \underline{a} \geq 0 \\ [\overline{a}^m, \underline{a}^m] & \overline{a} < 0 \\ [0, \max\{\underline{a}^m, \overline{a}^m\}] & \text{otherwise.} \end{cases} \tag{29}$$

Note that when m is odd A^m can be computed in terms of successive multiplications, which equals to state that $A^m = [\underline{a}, \overline{a}]^m = [\underline{a}^m, \overline{a}^m]$.

5 Preliminary Experiments

In order to use sign consistency as an effective means to support reasoning on polynomial constraints over finite domains, a prototype implementation of a specific algorithm has been recently included in the PolyFD library, which is a C++ library for constraint satisfaction still at an early development stage. The implemented algorithm enforces sign consistency on a constraint for a given bounding function by removing sign-inconsistent values from the domains of variables using a variant of the AC-3 algorithm [12]. The definition of sign consistency ensures that removed values are not included in any assignment that satisfies the constraint, and therefore the implemented algorithm can be used to support constraint satisfaction. The computational cost needed to enforce sign consistency using the implemented algorithm and the effectiveness of the algorithm to support constraint satisfaction largely depend on the used bounding function. Note that the current implementation of the PolyFD library is available upon request from authors, but it is not yet openly available to the research community because it is still at an early development stage.

The remaining of this section is devoted to present (in alphabetical order) five experiments on the use of the proposed bounding function to reason on well-known constraint satisfaction problems. In order to assess the effectiveness of the proposed approach, each problem was solved with the current prototype of the PolyFD library and with Gecode [19], and the results were compared in terms of execution time. In detail, each problem was solved 10 times using PolyFD and the shortest execution time t_P was recorded. Then, each problem was solved 10 times using Gecode and the shortest execution time t_G was recorded. Finally, t_P and t_G were compared. Note that both Gecode and PolyFD were configured to use the same search heuristics and, for a fair comparison, some of the features that Gecode implements were disabled because they are not yet implemented in PolyFD. For example, even if some of the problems require that all variables are assigned to different values, the specific filtering algorithm for the all-different global constraint [15] that Gecode implements was not used. Finally, note that Gecode has an established reputation for the advanced optimizations that it implements, so the results of experiments are considered satisfactory to assess the validity of the proposed bounding function to solve constraint satisfaction problems if t_P and t_G are within the same order of magnitude.

The following experiments where performed using Gecode (version 5.1.0) and PolyFD (version 2019.02.28) on a MacBook Air equipped with a 1.7 GHz Intel Core i5 processor and with 4 GB of memory. The experimented Gecode programs were based on available programs (`http://www.hakank.org/gecode`) with the addition of minor changes. The introduced changes were intended to remove the features of Gecode that are not yet implemented in PolyFD, and to ensure that studied problems have only one solution.

Example 1 (Corner). This example uses eight integer variables whose domains are initially set to interval [1..8]. Variables are enforced to be assigned to all different values, and the following six additional constraints are imposed

$$\begin{aligned} x_1 &= 1 & x_4 &= x_1 + x_6 \\ x_2 &= 4 & x_5 &= x_3 + x_8 \\ x_2 &= x_1 + x_3 & x_7 &= x_6 + x_8. \end{aligned}$$

The time t_G needed by the Gecode implementation to find the single solution to the problem, and the time t_P needed by the PolyFD implementation to find the same solution are

$$t_G = 0.651 \text{ ms} \qquad t_P = 0.647 \text{ ms}.$$

Note that PolyFD is roughly as fast as Gecode in finding the single solution to the studied problem.

Example 2 (Dinner). This example uses three integer variables whose domains are initially set to interval [1..100]. The following two constraints are imposed on variables

$$\begin{aligned} 6x_1 + 4x_2 + x_3 &= 40 \\ x_1 + x_2 + x_3 &= 20. \end{aligned}$$

The time t_G needed by the Gecode implementation to find the single solution to the problem, and the time t_P needed by the PolyFD implementation to find the same solution are

$$t_G = 0.544 \text{ ms} \qquad t_P = 0.640 \text{ ms}.$$

Note that PolyFD is 1.176 times slower than Gecode in finding the single solution to the studied problem.

Example 3 (Donald). This example uses ten integer variables whose domains are initially set to interval [0..9]. Variables are enforced to be assigned to all different values, it is requested that $x_1 \neq 0$, $x_6 \neq 0$, and $x_8 \neq 0$, and the following three additional constraints are imposed on variables

$$\begin{aligned} y_1 &= 100000x_1 + 10000x_2 + 1000x_3 + 100x_4 + 10x_5 + x_1 \\ y_2 &= 100000x_6 + 10000x_7 + 1000x_8 + 100x_4 + 10x_5 + x_1 \\ y_1 + y_2 &= 100000x_8 + 10000x_2 + 1000x_9 + 100x_7 + 10x_8 + x_{10}, \end{aligned}$$

where variables y_1 and y_2 are introduced for the sake of clarity but they are not part of the problem. The time t_G needed by the Gecode implementation to find the single solution to the problem, and the time t_P needed by the PolyFD implementation to find the same solution are

$$t_G = 1.152 \text{ ms} \qquad t_P = 1.112 \text{ ms}.$$

Note that PolyFD is 1.035 times faster than Gecode in finding the single solution to the studied problem.

Example 4 (Grocery). This example uses four integer variables whose domains are initially set to interval [0..711]. It is requested that $x_1 \leq x_2$, $x_2 \leq x_3$, and $x_3 \leq x_4$, and the following two additional constraints are imposed on variables

$$\begin{aligned} x_1 \cdot x_2 \cdot x_3 \cdot x_4 &= 711 \cdot 10^6 \\ x_1 + x_2 + x_3 + x_4 &= 711. \end{aligned}$$

The time t_G needed by the Gecode implementation to find the single solution to the problem, and the time t_P needed by the PolyFD implementation to find the same solution are

$$t_G = 168.002 \text{ ms} \qquad t_P = 32.635 \text{ ms}.$$

Note that PolyFD is 5.147 times faster than Gecode in finding the single solution to the studied problem.

Example 5 (Safe). This example uses nine integer variables whose domains are initially set to interval [1..9]. Variables are enforced to be all different, and the following thirteen additional constraints are imposed on variables

$$\begin{aligned} x_1 &\neq 1 \\ x_2 &\neq 2 \\ &\vdots \\ x_9 &\neq 9 \\ x_8 &> x_9 \\ x_7 &= x_4 - x_6 \\ x_1 \cdot x_2 \cdot x_3 &= x_8 + x_9 \\ x_2 + x_3 + x_6 &< x_8. \end{aligned}$$

The time t_G needed by the Gecode implementation to find the single solution to the problem, and the time t_P needed by the PolyFD implementation to find the same solution are

$$t_G = 0.863 \text{ ms} \qquad t_P = 1.654 \text{ ms}.$$

Note that PolyFD is 1.916 times slower than Gecode in finding the single solution to the studied problem.

6 Conclusion

This paper introduced and studied a bounding function based on interval arithmetic that is intended to make sign consistency a viable tool to solve practical constraint satisfaction problems. The simplicity and the low computational cost of the proposed bounding function make it a feasible option to treat practical constraint satisfaction problems, even if the bounds that it provides are less stringent than the bounds provided by other bounding functions.

In order to assess the applicability of the proposed bounding function to practical problems, preliminary experiments were performed to compare the performance of PolyFD, a prototype C++ library that uses the proposed bounding function, with the performance of Gecode. Notably, even if the programs that use PolyFD are sometimes slower than the corresponding programs that use Gecode, measured execution times are always within the same order of magnitude. In particular, in the worst case, the program that uses PolyFD is roughly twice as slow as the corresponding program that uses Gecode. On the contrary, in the best case, the program that uses PolyFD is more than five times faster the corresponding program that uses Gecode. Given that Gecode has an established reputation for the advanced level of optimizations that it implements, despite being preliminary, obtained experimental results can be considered encouraging.

A planned development of the presented work regards the addition of further experiments to the list of studied problems to improve the understanding of the feasibility of the proposed bounding function to reason on practical constraint satisfaction problems. Then, some optimizations on the implementation of the proposed bounding function have already been planned to take advantage of the sparsity patterns of polynomials. Given that the effectiveness of the implemented algorithm for sign-consistency enforcement largely depends on the computational cost of the adopted bounding function, improvements in the implementation of the proposed bounding function are expected to affect performance significantly. Finally, the improvement of the overall quality of the current implementation of PolyFD has already been planned in order to make it openly available to the research community of potentially interested users.

References

1. Apt, K.: Principles of Constraint Programming. Cambridge University Press (2003)
2. Bergenti, F., Monica, S.: Hyper-arc consistency of polynomial constraints over finite domains using the modified Bernstein form. Annals of Mathematics and Artificial Intelligence **80**(2), 131–151 (2017)
3. Bergenti, F., Monica, S.: Satisfaction of polynomial constraints over finite domains using function values. In: Della Monica, D., Murano, A., Rubin, S., Sauro, L. (eds.) ICTCS 2017 and CILC 2017 Italian Conference on Theoretical Computer Science and Italian Conference on Computational Logic. CEUR Workshop Proceedings, vol. 1949, pp. 262–275. RWTH Aachen (2017)
4. Bergenti, F., Monica, S., Rossi, G.: Polynomial constraint solving over finite domains with the modified Bernstein form. In: Fiorentini, C., Momigliano, A. (eds.)

CILC 2016 Italian Conference on Computational Logic. CEUR Workshop Proceedings, vol. 1645, pp. 118–131. RWTH Aachen (2016)
5. Bergenti, F., Monica, S., Rossi, G.: A subdivision approach to the solution of polynomial constraints over finite domains using the modified Bernstein form. In: Adorni, G., Cagnoni, S., Gori, M., Maratea, M. (eds.) AI*IA 2016 Advances in Artificial Intelligence. Lecture Notes in Computer Science, vol. 10037, pp. 179–191. Springer (2016)
6. Bergenti, F., Monica, S., Rossi, G.: Constraint logic programming with polynomial constraints over finite domains. Fundamenta Informaticae **161**(1–2), 9–27 (2018)
7. Farouki, R.T.: The Bernstein polynomial basis: A centennial retrospective. Computer Aided Geometric Design **29**(6), 379–419 (2012)
8. Farouki, R.T., Rajan, V.T.: Algorithms for polynomials in Bernstein form. Computer-Aided Geometric Design **5**(1), 1–26 (1988)
9. Garloff, J.: Convergent bounds for the range of multivariate polynomials. In: Nickel, K. (ed.) Interval Mathematics 1985. Lecture Notes in Computer Science, vol. 212, pp. 37–56. Springer (1986)
10. Garloff, J., Smith, A.P.: Solution of systems of polynomial equations by using Bernstein expansion. In: Alefeld, G., Rohn, J., Rump, S., Yamamoto, T. (eds.) Symbolic Algebraic Methods and Verification Methods. pp. 87–97. Springer (2001)
11. von zur Gathen, J., Gerhard, J.: Modern computer algebra. Cambridge University Press (2003)
12. Mackworth, A.: Consistency in networks of relations. Artificial Intelligence **8**, 99–118 (1977)
13. Ray, S., Nataraj, P.: An efficient algorithm for range computation of polynomials using the Bernstein form. Journal of Global Optimization **45**, 403–426 (2009)
14. Ray, S., Nataraj, P.: A matrix method for efficient computation of Bernstein coefficients. Reliable Computing **17**, 40–71 (2012)
15. Régin, J.C.: A filtering algorithm for constraints of difference in CSPs. In: Proceedings of the National Conference on Artificial Intelligence (AAAI 1994). pp. 362–367. AAAI (1994)
16. Rivlin, T.J.: Bounds on a polynomial. Journal of Research of the National Bureau of Standards **74B**(1), 47–54 (1970)
17. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming. Elsevier, New York (2006)
18. Ruszczyński, A.: Nonlinear Optimization. Princeton University Press (2006)
19. Schulte, C., Stuckey, P.J.: Efficient constraint propagation engines. ACM Transactions on Programming Languages and Systems **31**(1), 1–43 (2008)
20. Smith, A.P.: Fast construction of constant bound functions for sparse polynomials. Journal of Global Optimization **43**(2), 445–458 (2009)
21. Tucker, W.: Validated numerics: A short introduction to rigorous computations. Princeton University Press (2011)