Satisfaction of polynomial constraints over finite domains using function values

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. This paper shows how the solutions of constraint satisfaction problems that involve only polynomial constraints over finite domains can be enumerated by computing the values of related polynomial functions at appropriate points. The proposed algorithm first transforms constraints, which are expressed as equalities, inequalities, and disequalities of polynomials with integer coefficients and integer variables, into a canonical form that uses only inequalities. Then, starting from a bounding box, which is supposed to be known, the algorithm recursively subdivides the box into disjoint boxes and it records boxes whose elements satisfy all constraints. The subdivision is driven by the study of the sign of polynomial functions over boxes, which is performed by means of a method that uses only the coefficients of polynomials and the values of functions at the corners of boxes.

Keywords: polynomial constraints over finite domains, subdivision algorithms, Taylor's theorem

1 Introduction

Polynomial constraints are ubiquitous, and they find relevant applications in virtually all fields of science and engineering (see, e.g., [5, 13] and related literature). In this paper we study a specific form of polynomial constraints, which we call *polynomial constraints over finite domains* [1], in the scope of constraint logic programming and constraint satisfaction problems. Polynomial constraints over finite domains are constraints expressed as equalities, inequalities, and disequalities of polynomials with integer coefficients whose variables take values from finite subsets of the integers. Normally, reasoning on polynomial constraints over finite domains assumes that an initial approximation of the domains of variables is available in terms of a suitable bounding box.

We have already presented results on the use of the properties of *Bernstein polynomials* [4, 12] to reason on constraints satisfaction problems involving only polynomial constraints over finite domains [2, 3]. We have also proposed an algorithm to enforce hyper-arc consistency of this type of constraints using a subdivision algorithm [1], which also uses the properties of Bernstein polynomials. In this paper, the subdivision framework that we adopted in previous studies is used to enumerate the solutions of constraint satisfaction problems that involve only polynomial constraints over finite domains by computing the values of related polynomial functions at appropriate points, with no use of Bernstein polynomials. In detail, the proposed algorithm first rewrites constraints, which are expressed as equalities, inequalities and disequalities of polynomials with integer coefficients and integer variables, into a canonical form that uses only inequalities. Then, starting from an initial approximation of the domains of variables, which is supposed to be known in term of a bounding box, the algorithm recursively subdivides the box and it records boxes whose elements satisfy all constraints. Thanks to the adopted canonical form of constraints, the subdivision is driven by the study of the sign of polynomial functions over boxes. Section 3 presents results that show how the computation of the values of a polynomial function at the corners of a box provides relevant information on the sign of the function over the box. In detail, Section 3 describes a sufficient condition for a polynomial function to be nonnegative over a box by using only the coefficients of the polynomial and the values of the function at the corners of the box, which is exactly what is needed to drive the subdivision. If the sufficient condition is met then the elements of the considered box satisfy all constraints, otherwise the box should be split into disjoint boxes and processed recursively.

This paper is organised as follows. Section 2 recalls adopted canonical form of constraints and it outlines the subdivision algorithm, which is based on the possibility of identifying boxes where polynomial functions are nonnegative. Section 3 discusses how the computation of the value of a polynomial function at the corners of a box provides a sufficient condition for the function to be nonnegative over the box. Section 4 presents illustrative examples meant to clarify the proposed algorithm. Finally, Section 5 concludes the paper and it outlines open questions and future research directions.

2 Reasoning on polynomial constraints over boxes

Subdivision algorithms have been used extensively to reason on polynomial constraints mostly because effective bounds on the range of polynomial functions can be computed using the properties of Bernstein polynomials (see, [6] for a survey of major results and a historical perspective). Starting from a given box, lower and upper bounds for a polynomial function over the box can be computed by using the properties of Bernstein polynomials. Such bounds give relevant information on the sign of the function over the box, and they can be used to decide whether to recursively subdivide the box or not. This approach, which is informally known as *Bernstein algorithm* [9], is very general, and it has been used, for example, to solve systems of polynomial equations [10, 14] and to support global optimisation [11, 15].

Besides applications in mixed-integer nonlinear optimisation [16], the use of the Bernstein algorithm has been mostly limited to reason on polynomials constraints whose variables take values from the reals. Only recently, the properties of Bernstein polynomials have been used to reason on constraints whose variables take values from the integers, as follows. First, constraints are rewritten into a canonical form, which turns a generic polynomial constraint with integer coefficient whose variables take values from the integers into an inequality. Given a polynomial constraint of the form

$$p(\mathbf{x}) \odot q(\mathbf{x}), \tag{1}$$

where $p(\mathbf{x})$ and $q(\mathbf{x})$ are polynomials with integer coefficients in $n \in \mathbb{N}$ variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, and $\odot \in \{=, \neq, <, \leq, >, \geq\}$, a semantically equivalent constraint of the form

$$r(\mathbf{x}) \geq 0, \tag{2}$$

where $r(\mathbf{x})$ is a polynomial with integer coefficients, can be written by applying simple rules detailed in [1]. The possibility of transforming a generic polynomial constraint (1) into a semantically equivalent constraint of the specific form (2) can be used to turn the problem of searching for the values of variables that satisfy (1) into the problem of searching for the values of variables that make the polynomial function $r(\mathbf{x})$ nonnegative. This problem can be effectively solved using subdivision under the assumption that an initial approximation of the domains of variables is available. Actually, under this assumption, in [2, 3] we used subdivision to study the satisfiability of polynomial constraints over finite domains, while in [1] we used subdivision to make polynomial constraints over finite domains hyper-arc consistent. In both cases, we adopted the properties of Bernstein polynomials to study the signs of polynomial functions over boxes. In detail, the use of Bernstein polynomials to express a polynomial function over a given box allows the immediate identification of lower and upper bounds for the function over the box. Such bounds can be used to evaluate the sign of the function over the box, which can be recursively subdivided into disjoint boxes if the sign of the function changes over the box. This method is particularly appealing when variables are restricted to take values from finite subsets of the integers, because such a choice guarantees that the method always terminates, and also that results can be computed over the integers with no approximations, provided that arbitrary precision integer arithmetic is available.

The subdivision algorithm that we have already adopted to reason on polynomial constraints over finite domains does not mandate the use of a specific algorithm to study the sign of polynomial functions over boxes, and it can be generalised to a framework to reason on polynomial constraints over finite domains. In order to obtain working algorithms, the framework needs to be completed with a means to decide if a polynomial function is nonnegative over a box. This can be accomplished by means of any algorithm capable of computing suitable lower and upper bounds for a given polynomial function over a given box. If the lower bound is nonnegative, then the polynomial function is nonnegative over the box. Conversely, if the upper bound is negative, then the polynomial function is negative over the box. Note that the trivial approach of evaluating the polynomial function at all points of the box, which is always finite, is surely possible, but we expect that the use of lower and upper bounds would be beneficial especially for large boxes.

Algorithm 1 Function ENUMERATE(.,.) takes a set of constrains C in canonical form and a box $B \neq \emptyset$ over which the satisfiability of constraints in C is studied, and it returns a subset of B whose elements satisfy all constraints.

```
 1: function ENUMERATE($C, B$)
 2: input $C$ : a set of constraints          ▷ current set of constraints in canonical form
 3: input $B$ : a box in $\mathbb{Z}^n$                    ▷ nonempty box currently analysed
 4: output $S \subseteq B$               ▷ subset of $B$ in which all constraints are satisfied
 5:     if $C = \emptyset$ then
 6:         $S \leftarrow B$
 7:     else
 8:         select $c \in C$ with $c = (p(\mathbf{x}) \geq 0)$
 9:         $(l, u) \leftarrow$ BOUNDS(p, B)
10:         if $u < 0$ then
11:             $S \leftarrow \emptyset$
12:         else if $l \geq 0$ then
13:             $S \leftarrow$ ENUMERATE($C \backslash \{c\}, B$)
14:         else
15:             select $x_j$ with $I_j = [\underline{x}_j..\overline{x}_j]$ for $B = [\underline{x}_1..\overline{x}_1] \times [\underline{x}_2..\overline{x}_2] \times \cdots \times [\underline{x}_n..\overline{x}_n]$
16:             if $\underline{x}_j = \overline{x}_j$ then
17:                 $q(\mathbf{x}) \leftarrow p(\mathbf{x})$ with $x_j$ replaced by $\overline{x}_j$
18:                 $S \leftarrow$ ENUMERATE($C \backslash \{c\} \cup \{q(\mathbf{x}) \geq 0\}, B$)
19:             else
20:                 select $s \in I_j$ with $s \neq \overline{x}_j$
21:                 $S' \leftarrow$ ENUMERATE($C, B_{j \to [\underline{x}_j..s]}$)
22:                 $S'' \leftarrow$ ENUMERATE($C, B_{j \to [s+1..\overline{x}_j]}$)
23:                 $S \leftarrow S' \cup S''$
24:             end if
25:         end if
26:     end if
27: end function
```

Algorithm 1 shows a pseudocode of the proposed framework to enumerate the solutions of constraint satisfaction problems that involve only polynomial constraints over finite domains. Function ENUMERATE(.,.) takes a set of constraints C in canonical form and a box $B \subset \mathbb{Z}^n$, with $B \neq \emptyset$, over which the signs of polynomials in C are studied. First, ENUMERATE(.,.) selects a constraint and it uses the unspecified function BOUNDS(.,.) to compute appropriate lower and upper bounds of the related polynomial function over B. If computed upper bound is negative, then the constraint is unsatisfiable in the box. Conversely, if computed lower bound is nonnegative, then the selected constraint is satisfiable for all elements of the box. If both conditions are not met, then a variable is selected and the current approximation of its domain is properly split into two disjoint intervals, which are processed recursively. The notation $B_{j \to [a..b]}$ is used to denote a box obtained from box B by replacing interval I_j, which corresponds to the current approximation of the domain of the $j-$th variable, with interval $[a..b]$. Note that the pseudocode in Algorithm 1 considers also the special case of variables whose domains are reduced to singleton sets. In this case, variables can be substituted in polynomials with appropriate values, so that the number of variables in polynomials is reduced. Finally, note that function ENUMERATE(.,.) is first invoked with the initial set of constraints and the initial approximation of the domains of variables, which is expressed as a nonempty bounding box.

3 Study of the sign of polynomials using function values

In this section we present a method to study the sign of a polynomial function over a given box that is based on a result originally proposed for univariate polynomials in [18], which was then extended to bivariate polynomials in [8]. The most general result, which holds for multivariate polynomials, is shown, but not proved for the sake of brevity, in Proposition 2. First, the case of univariate polynomials is discussed to explain the proposed method in a simple setting. Then, the proposed extension to multivariate polynomials is shown using the elegant notation of multi-indices. Such an extension can be used as a basis to implement function BOUNDS(.,.) of the pseudocode in Algorithm 1.

3.1 Univariate polynomials

Let $p : \mathbb{R} \mapsto \mathbb{R}$ be a polynomial function, and $B \subset \mathbb{R}$ be a closed interval. We are interested in computing a lower bound $\underline{m} \in \mathbb{R}$ and an upper bound $\overline{m} \in \mathbb{R}$ of p over B such that

$$\underline{m} \leq \min_{x \in B} p(x) \qquad \max_{x \in B} p(x) \leq \overline{m}. \tag{3}$$

In order to compute $\underline{m}$ and $\overline{m}$, we recall the Taylor's theorem for functions in one variable, which is rephrased here to fix notation and because there is little consensus on what should be called Taylor's theorem. The theorem is classic and it is shown without proof.

Theorem 1 (Taylor's theorem in one variable). *Let $f : \mathbb{R} \mapsto \mathbb{R}$ be of class $C^{(k+1)}$, with $k \in \mathbb{N}$, on an open interval $S \subseteq \mathbb{R}$. If $a \in S$, $h \in \mathbb{R}$ and $a + h \in S$, then*

$$f(a+h) = \sum_{i=0}^{k} \frac{D^{(i)} f(a)}{i!} h^i + R_{a,k}(h), \tag{4}$$

where $D^{(i)} f$ is the $i-$th derivative of f (with $D^{(0)} f = f$), and the remainder $R_{a,k}(h)$ is given in Lagrange form by

$$R_{a,k}(h) = \frac{D^{(k+1)} f(a + ch)}{(k+1)!} h^{(k+1)} \tag{5}$$

for some $c \in (0, 1)$.

The application of the Taylor's theorem to a polynomial function on a closed interval easily proves the following corollary.

Corollary 1. *Let $p : \mathbb{R} \mapsto \mathbb{R}$ be a polynomial function and $B = [\underline{b}, \overline{b}] \subset \mathbb{R}$ be a closed interval. Then, for all $x \in B$, and for all $k \in \mathbb{N}$*

$$p(x) = \sum_{i=0}^{k} \frac{D^{(i)}p(\underline{b})}{i!}(x - \underline{b})^i + R_{\underline{b},k}(x - \underline{b}), \tag{6}$$

where the remainder $R_{\underline{b},k}(h)$ is given in Lagrange form by

$$R_{\underline{b},k}(h) = \frac{D^{(k+1)}p(\underline{b} + ch)}{(k+1)!}h^{(k+1)} \tag{7}$$

for some $c \in (0, 1)$.

Proof. Recalling that any polynomial function is of class C^∞ on any open interval $S \subseteq \mathbb{R}$ with $B \subset S$, the proof of the corollary follows immediately from the Taylor's theorem. $\square$

In particular, if we truncate to $k = 2$, we have that

$$p(x) = p(\underline{b}) + D^{(1)}p(\underline{b})(x - \underline{b}) + \frac{D^{(2)}p(\underline{b} + c(x - \underline{b}))}{2}(x - \underline{b})^2 \tag{8}$$

for some $c \in (0, 1)$. The truncated expansion (8) helps to prove the following proposition (see also [18]).

Proposition 1. *Let $p : \mathbb{R} \mapsto \mathbb{R}$ be a polynomial function of degree $l \in \mathbb{N}$*

$$p(x) = \sum_{i=0}^{l} a_i x^i \tag{9}$$

with coefficients $\{a_i\}_{i=0}^{l} \subset \mathbb{R}$, and let $U = [0, 1] \subset \mathbb{R}$ with

$$\underline{m} = \min_{x \in U} p(x) \qquad \overline{m} = \max_{x \in U} p(x). \tag{10}$$

Then

$$\min_{x \in \{0,1\}} p(x) - \delta \leq \underline{m} \qquad \overline{m} \leq \max_{x \in \{0,1\}} p(x) + \delta, \tag{11}$$

where

$$\delta = \frac{1}{8} \sum_{i=2}^{l} (i - 1)i|a_i|. \tag{12}$$

Proof. Let $\eta \in U$ be such that $p(\eta) = \underline{m}$, then if $\eta = 0$ or $\eta = 1$ the proposition is immediately proved because $\delta \geq 0$ and therefore

$$\min_{x \in \{0,1\}} p(x) - \delta \leq \underline{m}. \tag{13}$$

Conversely, if $\eta \neq 0$ and $\eta \neq 1$, then $D^{(1)}p(\eta) = 0$. In this case, using the fact that the choice of interval U ensures that

$$0 \leq x^i \leq 1, \tag{14}$$

with $i \in \mathbb{N}$, and $x \in U$, the expansion of the second derivative in (8) is sufficient to prove the proposition. In detail, if $\eta \leq \frac{1}{2}$, then

$$\underline{m} = p(0) + \frac{D^{(2)}p(c\eta)}{2}\eta^2 \tag{15}$$

for some $c \in (0, 1)$. But,

$$D^{(2)}p(x) = \sum_{i=2}^{l} i(i-1)a_i x^{(i-2)} \leq \sum_{i=2}^{l} i(i-1)|a_i|, \tag{16}$$

for all $x \in U$, which proves the proposition for $\eta \leq \frac{1}{2}$. For the case $\eta > \frac{1}{2}$, the proposition is proved by choosing the other endpoint of interval U to expand the function, which leads to an equality similar to (15). The case of $\overline{m}$ is treated with similar considerations. □

Proposition 1 ensures that, for the case of univariate polynomials, suitable bounds to support the pseudocode in Algorithm 1 can be obtained by computing the values of polynomial functions at the endpoints of considered intervals. Note that the assumption $x \in U$ in Proposition 1 is not restrictive because, in order to consider a polynomial function $p : \mathbb{R} \mapsto \mathbb{R}$ in variable x which takes values from interval $B = [\underline{b}, \overline{b}] \subset \mathbb{R}$, it is sufficient to change the variable of p to $t \in U$ such that

$$x = (\overline{b} - \underline{b})t + \underline{b}, \tag{17}$$

and to consider the coefficient of the new polynomial in variable t. In this case, we prefer to use the notation δ_B in (12), instead of simply δ, because the coefficients of the new polynomial make δ_B depend on B.

3.2 Multivariate polynomials

In order to elegantly generalise Proposition 1 to the case of polynomials with $n \in \mathbb{N}$ variables, we need to introduce the notation of multi-indices. A multi-index $I \in \mathbb{N}^n$ is an n-tuple of the form

$$I = (i_1, i_2, \ldots, i_n), \tag{18}$$

and we adopt the common notation of using an uppercase letter to denote a multi-index and the same letter, but lowercase and with an index, to denote its components. Given a multi-index $I \in \mathbb{N}^n$, the following abbreviations

$$|I| = \sum_{j=1}^{n} i_j \qquad I! = \prod_{j=1}^{n} i_j! \tag{19}$$

are often used to denote the *order* of I, and the *factorial* of I, respectively. Given two multi-indices $I \in \mathbb{N}^n$ and $J \in \mathbb{N}^n$, they can be added

$$I + J = (i_1 + j_1, i_2 + j_2, \ldots, i_n + j_n), \tag{20}$$

and they can be compared

$$I \leq J \iff i_1 \leq j_2 \wedge i_2 \leq j_2 \wedge \cdots \wedge i_n \leq j_n, \tag{21}$$

which justify the following abbreviations, with $k \in \mathbb{N}$,

$$\sum_{I \leq J} (\cdot) = \sum_{i_1=0}^{j_1} \sum_{i_2=0}^{j_2} \cdots \sum_{i_n=0}^{j_n} (\cdot) \tag{22}$$

$$\sum_{|I| \leq k} (\cdot) = \underbrace{\sum_{i_1=0}^{k} \sum_{i_2=0}^{k} \cdots \sum_{i_n=0}^{k}}_{i_1+i_2+\cdots+i_n \leq k} (\cdot)\,. \tag{23}$$

Given a multi-index $I \in \mathbb{N}^n$ and $\mathbf{v} \in \mathbb{R}^n$, the following abbreviation is often used

$$\mathbf{v}^I = \prod_{j=0}^{n} v_j^{i_j}. \tag{24}$$

Finally, given a generic function $f : \mathbb{R}^n \mapsto \mathbb{R}$ in variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, the following notation based on multi-indices is used to denote elegantly the partial derivatives of f

$$\partial^I f = \partial^{i_1} \partial^{i_2} \cdots \partial^{i_n} f = \frac{\partial^{|I|} f}{\partial {x_1}^{i_1} \partial {x_2}^{i_2} \cdots \partial {x_n}^{i_n}}. \tag{25}$$

The notation of multi-indices can be used to state the following theorem, which is then used to study the sign of a polynomial function over a given box by computing the value of the function at the corners of the box. The theorem generalises the Taylor's theorem recalled above, and it is shown without proof.

Theorem 2 (Taylor's theorem in several variables). *Let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be of class $C^{(k+1)}$, with $k \in \mathbb{N}$, on an open convex $S \subseteq \mathbb{R}^n$. If $\mathbf{a} \in S$, $\mathbf{h} \in \mathbb{R}^n$ and $\mathbf{a} + \mathbf{h} \in S$, then*

$$f(\mathbf{a} + \mathbf{h}) = \sum_{|I| \leq k} \frac{\partial^I f(\mathbf{a})}{I!} \mathbf{h}^I + R_{\mathbf{a},k}(\mathbf{h}), \tag{26}$$

where the remainder $R_{\mathbf{a},k}(\mathbf{h})$ is given in Lagrange form by

$$R_{\mathbf{a},k}(\mathbf{h}) = \sum_{|I|=k+1} \frac{\partial^I f(\mathbf{a} + c\mathbf{h})}{I!} \mathbf{h}^I \tag{27}$$

for some $c \in (0, 1)$.

Let $p : \mathbb{R}^n \mapsto \mathbb{R}$ be a multivariate polynomial function in $n \in \mathbb{N}$ variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, the adoption of multi-index notation allows expressing p as

$$p(\mathbf{x}) = \sum_{I \leq L} a_I \mathbf{x}^I, \tag{28}$$

where $L \in \mathbb{N}^n$ is the *multi-degree* of the polynomial function, and $\{a_I\}_{I \leq L} \subset \mathbb{R}$ is the set of its coefficients.

Given that we are interested in studying the sign of polynomial functions in boxes, we introduce the following notation to denote boxes. The box B built using $\underline{\mathbf{v}} = (\underline{v}_1, \underline{v}_2, \ldots, \underline{v}_n) \in R^n$ and $\overline{\mathbf{v}} = (\overline{v}_1, \overline{v}_2, \ldots, \overline{v}_n) \in R^n$ is denoted as

$$B = [\underline{\mathbf{v}}, \overline{\mathbf{v}}] = [\underline{v}_1, \overline{v}_1] \times [\underline{v}_2, \overline{v}_2] \times \cdots \times [\underline{v}_n, \overline{v}_n], \tag{29}$$

and it is the empty set if and only if for some $0 \leq i \leq n$, $\underline{v}_i > \overline{v}_i$.

Adopted notations can be used to state a generalisation of Corollary 1 for multivariate polynomials.

Corollary 2. *Let $p : \mathbb{R}^n \mapsto \mathbb{R}$ be a polynomial function in $n \in \mathbb{N}$ variables, and $B = [\underline{\mathbf{b}}, \overline{\mathbf{b}}] \subset \mathbb{R}^n$ be a box. Then, for all $\mathbf{x} \in B$, and for all $k \in \mathbb{N}$*

$$p(\mathbf{x}) = \sum_{|I| \leq k} \frac{\partial^I p(\underline{\mathbf{b}})}{I!} (\mathbf{x} - \underline{\mathbf{b}})^I + R_{\underline{\mathbf{b}},k}(\mathbf{x} - \underline{\mathbf{b}}) \tag{30}$$

and the remainder $R_{\underline{\mathbf{b}},k}(\mathbf{h})$ is given in Lagrange form by

$$R_{\underline{\mathbf{b}},k}(\mathbf{h}) = \sum_{|I| = k+1} \frac{\partial^I p(\underline{\mathbf{b}} + c\mathbf{h})}{I!} \mathbf{h}^I \tag{31}$$

for some $c \in (0, 1)$.

Proof. Recalling that any polynomial function is of class C^∞ for any open convex $S \subseteq \mathbb{R}^n$ with $B \subset S$, the proof of the corollary follows immediately from the Taylor's theorem. $\square$

Following the same path traced for the case of univariate polynomials, we can prove the following proposition, which is sufficient to compute appropriate lower and upper bounds to support the proposed algorithm. The proposition is shown without proof for the sake of brevity.

Proposition 2. *Let $p : \mathbb{R}^n \mapsto \mathbb{R}$ be a polynomial function with multi-degree $L \in \mathbb{N}$ in $n \in \mathbb{N}$ variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$*

$$p(\mathbf{x}) = \sum_{I \leq L} a_I \mathbf{x}^I \tag{32}$$

and $U = [0, 1]^n \subset \mathbb{R}^n$, with

$$\underline{m} = \min_{\mathbf{x} \in U} p(\mathbf{x}) \qquad \overline{m} = \max_{\mathbf{x} \in U} p(\mathbf{x}). \tag{33}$$

Then

$$\min_{\mathbf{x}\in C} p(\mathbf{x}) - \delta \leq \underline{m} \qquad \overline{m} \leq \max_{\mathbf{x}\in C} p(\mathbf{x}) + \delta, \tag{34}$$

where $C = \{0,1\}^n$ *is the set of the corners of* U *and*

$$\delta = \frac{1}{8} \sum_{I \leq L} |I|(|I|-1)|a_I|. \tag{35}$$

Note that the assumption $\mathbf{x} \in U$ in Proposition 2 is not restrictive. In order to consider a polynomial function $p : \mathbb{R}^n \mapsto \mathbb{R}$ in variable $\mathbf{x}$ which takes values from box $B = [\underline{\mathbf{b}}, \overline{\mathbf{b}}] \subset \mathbb{R}^n$, it is sufficient to change the variable of p to $\mathbf{t} \in U$ such that

$$\mathbf{x} = (\overline{\mathbf{b}} - \underline{\mathbf{b}})\mathbf{t} + \underline{\mathbf{b}}, \tag{36}$$

and to consider the coefficient of the new polynomial in variable $\mathbf{t}$. In this case, we prefer to use the notation δ_B in (35), instead of simply δ, because the coefficients of the new polynomial make δ_B depend on B.

4 Illustrative examples

This section shows how the proposed algorithm can be applied to some illustrative examples. Note that a systematic validation of the proposed algorithm with specific benchmarks is still in progress, and the following examples are only intended to clarify the algorithm. The first example is a quadratic constraint with only one variable

$$p_1(x) = -x^2 - 50 \geq 0 \qquad x \in [1..25]. \tag{37}$$

The constraint is unsatisfiable in the given interval, and the proposed algorithm proves it by subdividing the initial interval into 4 intervals, the largest of which contains 12 values. Figure 1 shows the subdivision produced by the algorithm.

The second example is an unsatisfiable constraint with two variables

$$p_2(x,y) = -x^2 - y^2 - 100 \geq 0 \qquad x \in [1..25],\ y \in [1..25]. \tag{38}$$

The proposed algorithm needs to subdivide the initial box into 12 boxes, the largest of which contains 144 values, to prove that the constraint is unsatisfiable. Figure 2 shows the subdivision of the box that the algorithm produces.

The following examples are relative to larger domains and the detailed subdivision of the initial approximations of domains are too complex to be shown. The following example is a satisfiable constraint with one variable

$$p_3(x) = x^2 - 16 \geq 0 \qquad x \in [-100..100]. \tag{39}$$

The proposed algorithm builds a search tree made of 65 nodes to compute the following answer

$$x \in [-100..-4] \cup [4..100]. \tag{40}$$

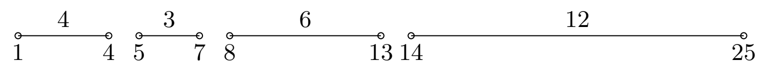

Fig. 1. Subdivision of interval $I = [1..25]$ that the proposed algorithm produces to prove that $p_1(x) = -x^2 - 50 \geq 0$ is unsatisfiable in I. The cardinality of each interval is also shown.

Given that each node requires the computation of the value of p_3 at the two endpoints of the related interval, 130 evaluations of p_3 are needed. Note that a brute force investigation requires 201 evaluations of the polynomial.

The fourth example is still with only one variable

$$p_4(x) = x^2 - 50x + 1 \geq 0 \qquad x \in [-100..100]. \tag{41}$$

In this case the answer

$$x \in [-100..0] \cup [50..100] \tag{42}$$

is returned after computing a search tree made of 61 nodes, which requires 122 evaluations of p_4, against the 201 evaluations needed by brute force investigation.

The fifth illustrative example involves two variables

$$p_5(x, y) = xy - 210 \geq 0 \qquad x \in [-100..100],\, y \in [-100..100]. \tag{43}$$

The proposed algorithm needs to traverse a search tree made of $1,909$ nodes to compute the answer

$$x \in [-100.. - 3] \cup [3..100] \qquad y \in [-100.. - 3] \cup [3..100]. \tag{44}$$

Note that the computation of the answer required $7,636$ evaluations of p_5, four for each considered box, while the brute force scan of the domains of variables requires $40,401$ evaluations of the polynomial.

Finally, the last example involves two variables

$$p_6(x, y) = x + xy - 1000 \geq 0 \qquad x \in [-100..100],\, y \in [-100..100]. \tag{45}$$

In this case the answer

$$x \in [-100.. - 11] \cup [10..100] \qquad y \in [-100.. - 11] \cup [9..100] \tag{46}$$

is computed after traversing a search tree made of $1,347$ nodes, which required $5,388$ evaluations of p_6, against the $40,401$ evaluations of brute force search.

Note that discussed examples are processed using the current implementation of the algorithm, which intentionally does not use heuristics to select which variable to split, and where to split the domain of the selected variable. It is expected that the introduction of heuristics to drive the search would bring notable performance improvements.

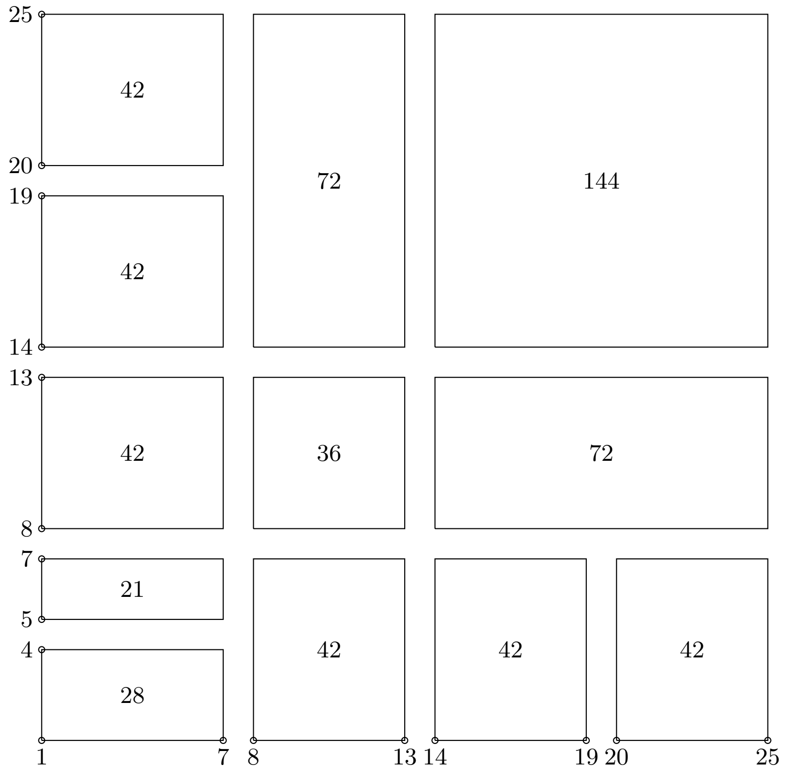

Fig. 2. Subdivision of box $B = [1..25] \times [1..25]$ that the proposed algorithm produces to prove that $p_2(x, y) = -x^2 - y^2 - 100 \geq 0$ is unsatisfiable in B. The cardinality of each box is also shown.

5 Conclusions

This paper presented an algorithm to enumerate the solutions of constraint satisfaction problems that involve only polynomial constraints over finite domains. Such constraints are expressed as equalities, inequalities, and disequalities of polynomials with integer coefficients whose variables take values from finite subsets of the integers. The proposed algorithm works under the assumption that an initial approximation of the domains of variables is available in terms of a bounding box. First, the algorithm rewrites all constraints into a canonical form in which only inequalities are used. Then, it recursively subdivides the initial bounding box into disjoint boxes, and it records boxes whose elements satisfy all constraints. The canonical form of constraints ensures that a box contains only elements that satisfy all constraints if and only if the polynomial functions related to constraints in canonical form are all nonnegative over the box. In order to verify if a polynomial function is nonnegative over a given box, the algorithm uses lower and upper bounds described in Proposition 2. Such bounds are computed by considering the coefficients of the polynomial function and its values at the corners of the considered box.

The algorithm was tested on illustrative examples and preliminary results are encouraging, but a systematic validation with specific benchmarks is still in progress. We expect that the proposed algorithm would outperform our previous proposals based on the properties of Bernstein polynomials because of the inherent complexity of computing Bernstein coefficients, even using sophisticated techniques (see, e.g., [7, 17, 19, 20]). Moreover, it is evident that the performance of the algorithm on single problems severely depends on the details of the subdivision, which are deliberately not considered in this paper. In particular, it is expected that nontrivial heuristics to decide which domain to split and where to split it would be largely beneficial in practice. Heuristics could use the specific form of treated constraints to reason on the next variable to select for subdivision, and on where to subdivide its domain, because the gradient of polynomial functions can be computed easily.

The proposed algorithm is implemented as a reusable Java library that can be accessed as a Prolog module from SWI-Prolog [22]. The current implementation of the Java library and of the interface to SWI-Prolog can be downloaded in a single package (`cmt.dmi.unipr.it/software/clppolyfd.zip`). To ease experimentation, the Prolog module is fully compatible with the CLP(FD) module [21] of SWI-Prolog, and it can be configured to use the proposed algorithm or the algorithm based on Bernstein polynomials discussed in [1–3].

References

1. 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)
2. Bergenti, F., Monica, S., Rossi, G.: Polynomial constraint solving over finite domains with the modified Bernstein form. In: Fiorentini, C., Momigliano, A. (eds.) 31st Italian Conference on Computational Logic. CEUR Workshop Proceedings, vol. 1645, pp. 118–131. RWTH Aachen (2016)
3. 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 International Publishing (2016)
4. Bernstein, S.N.: Démonstration du théorème de Weierstrass fondée sur le calcul des probabilités. Communications de la Société Mathématique de Kharkov 2:XIII(1), 1–2 (1912)
5. Borralleras, C., Lucas, S., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: SAT modulo linear arithmetic for solving polynomial constraints. Journal of Automated Reasoning 48(1), 107–131 (2010)
6. Farouki, R.T.: The Bernstein polynomial basis: A centennial retrospective. Computer-Aided Geometric Design 29(6), 379–419 (2012)

7. Farouki, R.T., Rajan, V.T.: Algorithms for polynomials in Bernstein form. Computer-Aided Geometric Design 5(1), 1–26 (1988)
8. 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 International Publishing (1986)
9. Garloff, J.: The Bernstein algorithm. Interval Computations 2, 154–168 (1993)
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 International Publishing (2001)
11. Grimstad, B., Sandnes, A.: Global optimization with spline constraints: A new branch-and-bound method based on B-splines. Journal of Global Optimization 65(3), 401–439 (2016)
12. Lorentz, G.G.: Bernstein Polynomials. University of Toronto Press, Toronto (1953)
13. Lucas, S., Navarro-Marset, R.: Comparing CSP and SAT solvers for polynomial constraints in termination provers. Electronic Notes in Theoretical Computer Science 206, 75–90 (2008)
14. Mourrain, B., Pavone, J.: Subdivision methods for solving polynomial equations. Journal of Symbolic Computation 44(3), 292–306 (2009)
15. Nataraj, P., Arounassalame, M.: A new subdivision algorithm for the Bernstein polynomial approach to global optimization. International Journal of Automation and Computing 4(4), 342–352 (2007)
16. Patil, B.V., Nataraj, P.S.V., Bhartiya, S.: Global optimization of mixed-integer nonlinear (polynomial) programming problems: The Bernstein polynomial approach. Computing 94(2), 325–343 (2012)
17. Ray, S., Nataraj, P.: An efficient algorithm for range computation of polynomials using the Bernstein form. Journal of Global Optimization 45, 403–426 (2009)
18. Rivlin, T.J.: Bounds on a polynomial. Journal of Research of the National Bureau of Standards 74B(1), 47–54 (1970)
19. Sánchez-Reyes, J.: Algebraic manipulation in the Bernstein form made simple via convolutions. Computer-Aided Design 35, 959–967 (2003)
20. Smith, A.P.: Fast construction of constant bound functions for sparse polynomials. Journal of Global Optimization 43(2), 445–458 (2009)
21. Triska, M.: The finite domain constraint solver of SWI-Prolog. In: Schrijvers, T., Thiemann, P. (eds.) Functional and Logic Programming. Lecture Notes in Computer Science, vol. 7294, pp. 307–316. Springer International Publishing (2012)
22. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theory and Practice of Logic Programming 12(1–2), 67–96 (2012)