Ann Math Artif Intell (2013) 67:359–382
DOI 10.1007/s10472-013-9365-7

Representing and solving finite-domain constraint problems using systems of polynomials

Christopher Jefferson · Peter Jeavons · Martin J. Green · M. R. C. van Dongen

Published online: 12 June 2013

Abstract In this paper we investigate the use of a system of multivariate polynomials to represent the restrictions imposed by a collection of constraints. One advantage of using polynomials to represent constraints is that it allows many different forms of constraints to be treated in a uniform way. Systems of polynomials have been widely studied, and a number of general techniques have been developed, including algorithms that generate an equivalent system with certain desirable properties, called a Gröbner Basis. General algorithms to compute a Gröbner Basis have doubly exponential complexity, but we observe that for the systems we use to describe constraint problems over finite domains a Gröbner Basis can be computed more efficiently than this. We also describe a family of algorithms, related to the calculation of Gröbner Bases, that can be used on any system of polynomials to compute an equivalent system in polynomial time. We show that these modified algorithms can simulate the effect of the local-consistency algorithms used in constraint programming and hence solve certain broad classes of constraint problems in

Christopher Jefferson was supported by EPSRC Grant EP/D032636/1 and Martin Green was supported by EPSRC Grant EP/C525949/1.

C. Jefferson (✉)
Department of Computer Science, University of St Andrews, St Andrews, UK
e-mail: caj@cs.st-andrews.ac.uk

P. Jeavons
Department of Computer Science, University of Oxford, Oxford, UK
e-mail: Peter.Jeavons@cs.ox.ac.uk

M. J. Green
Department of Computer Science, Royal Holloway, University of London, Egham, UK
e-mail: M.J.Green@cs.rhul.ac.uk

M. R. C. van Dongen
Department of Computer Science, University College Cork, Cork, Ireland
e-mail: Dongen@cs.ucc.ie

polynomial time. Finally we discuss the use of adaptive consistency techniques for systems of polynomials.

Keywords Constraint satisfaction · Groebner basis · Consistency

Mathematics Subject Classifications (2010) 68R05 · 13P10 · 13P25

1 Introduction

The constraint programming paradigm [1] involves modelling a real-world problem as a set of *variables* together with a set of *constraints*; the constraints restrict the allowed combinations of values that can be simultaneously assigned to the variables. As well as capturing many practical computational problems, this very general paradigm includes a number of special cases corresponding to problems with particular forms of constraints, such as Integer Programming and Satisfiability. The study of efficient ways to represent and solve such constraint problems is currently a very active area of research within artificial intelligence [1].

In a constraint programming system each variable has to be assigned a value from some specified (often finite) domain, and each constraint describes the set of allowed combinations of values for some subset of the variables. A constraint may sometimes be specified *explicitly*, by listing the allowed (or disallowed) combinations of values in a table. More often, a constraint is specified *implicitly* by using some combination of available predicates, equations, inequalities and logical connectives.

In this paper we explore the use of *systems of polynomials* to specify constraints. We will say that a system of polynomials *allows* a particular combination of values for a set of variables if the simultaneous assignment of those values to the variables makes all of the polynomials in the system evaluate to zero. Such an assignment is called a *solution* to the system of polynomials.

The use of systems of polynomials has been considered a number of times in the constraints literature [2–5], but is typically used to represent constraints on continuous variables. Here we focus on the use of polynomials to represent finite-domain constraints. One advantage of representing such constraints using polynomials is that they can then be treated in a uniform way along with continuous constraints, allowing the development of very general constraint-solving techniques. Another advantage is that systems of polynomials can be processed by standard computer algebra packages such as Mathematica and REDUCE, so our approach helps to unify constraint programming with other forms of mathematical programming.

The approach of using polynomials to represent finite-domain constraints has previously been explored by Clegg, Edmonds and Impagliazzo, but only for the special case of Satisfiability problems [6]. Early work on this special case of Boolean problems was also reported in [7] (and surveyed more recently in [8]). A rather different approach to using polynomials to represent the specific linear constraints arising in integer programming was suggested in [9] and extended by [10].

Polynomials provide a very flexible and generic representation for problems from a wide variety of areas, and they have been intensively studied. In particular, Buchberger [11–13] showed that for any system of polynomials it is possible to compute an equivalent system with a number of desirable properties, which he called

a *Gröbner Basis*. Given a Gröbner Basis, it is possible to obtain the solutions to a system of polynomials very easily (or determine that it has no solutions). A Gröbner Basis provides a convenient representation for the whole set of solutions which can be used to answer a wide range of questions about them, such as the correlations between individual variables, or the total number of solutions.

To make this paper more self-contained, and accessible to constraint programmers, we give a brief overview of polynomials and standard Gröbner Basis techniques in Sections 2 and 4. In general, the complexity of computing a Gröbner Basis for a system of polynomials is doubly exponential in the size of the system, except in certain special cases. However, we observe in Section 5 that for the systems we use to represent constraints over finite domains this complexity is only singly exponential, and so is comparable with other general search techniques for constraints. Our main technical contributions are in Sections 6 and 7. In Section 6 we discuss a modified form of the algorithm for computing a Gröbner Basis which runs in *polynomial* time, but is incomplete, in the sense that it computes a system which is not necessarily a Gröbner Basis. The system computed by our modified algorithm has the same solutions as the original system and represents a set of equivalent constraints which is *locally consistent*. We show in Section 7 that this modified form of the algorithm can be used to achieve the same benefits as the local-consistency algorithms which are widely used in constraint processing. Finally, in Section 8 we discuss the use of adaptive consistency techniques for systems of polynomials.

2 Polynomials, varieties and ideals

In this section we introduce the basic concepts related to polynomials which are used throughout the paper. This section provides a brief tutorial introduction aimed at researchers in constraint programming. For a more detailed introduction see, for example, [14].

Definition 1 Let $\mathbb{K}$ be any field and $\{x_1, \ldots, x_n\}$ a set of variables. A **polynomial** $p(x_1, \ldots, x_n)$ over $\mathbb{K}$ on the variables $x_1, \ldots, x_n$ is a finite sum of **terms** of the form $\kappa {x_1}^{\alpha_1} \cdots {x_n}^{\alpha_n}$, where $\kappa \in \mathbb{K}$ and $\alpha_i \in \{0, 1, 2, \ldots\}, i = 1, \ldots, n$. A **monomial**[1] is a single product ${x_1}^{\alpha_1} \cdots {x_n}^{\alpha_n}$ over $\{x_1, \ldots, x_n\}$.

The set of all polynomials over $\mathbb{K}$ with variables $x_1, \ldots, x_n$, together with the standard addition and multiplication operations, forms a ring, which is denoted $\mathbb{K}[x_1, \ldots, x_n]$. A **system of polynomials** is a set of polynomials which all belong to the same polynomial ring; this set may be finite or infinite.

Definition 2 Let P be a system of polynomials from the ring $\mathbb{K}[x_1, \ldots, x_n]$. The **variety** of P, denoted $\mathrm{V}(P)$, is the set of all solutions to P, that is:

$$\mathrm{V}(P) = \left\{ \langle \kappa_1, \ldots, \kappa_n \rangle \in \mathbb{K}^n : \forall p \in P,\ p(\kappa_1, \ldots, \kappa_n) = 0 \right\}.$$

[1] Unfortunately, the words "term" and "monomial" are used by some authors the opposite way round! Here we follow the terminology of [14].

Definition 3 An **ideal** I over the polynomial ring $\mathbb{K}[x_1, \ldots, x_n]$ is a system of polynomials from $\mathbb{K}[x_1, \ldots, x_n]$ which satisfies the following conditions:

1. For all $f_1, f_2 \in I$ we have that $f_1 + f_2 \in I$;
2. For all $u \in \mathbb{K}[x_1, \ldots, x_n]$ and all $f \in I$ we have that $uf \in I$.

Definition 4 Let $\{p_1, \ldots, p_m\}$ be a finite system of polynomials from the ring $\mathbb{K}[x_1, \ldots, x_n]$. The **ideal generated by** $\{p_1, \ldots, p_m\}$, denoted $\mathrm{I}(\{p_1, \ldots, p_m\})$, is given by

$$\mathrm{I}(\{p_1, \ldots, p_m\}) = \left\{ \sum_{i=1}^{m} u_i p_i \ : \ u_i \in \mathbb{K}[x_1, \ldots, x_n] \right\} .$$

Given a finite system of polynomials $P = \{p_1, \ldots, p_m\}$, from $\mathbb{K}[x_1, \ldots, x_n]$, it is easy to see that the ideal generated by P has exactly the same set of solutions as the original system, that is, $\mathrm{V}(\mathrm{I}(P)) = \mathrm{V}(P)$.

Example 1 In this example, assume the polynomials are drawn from $\mathbb{C}[x, y, z]$, where $\mathbb{C}$ is the field of complex numbers.

The ideal $\mathrm{I}(\{x + y, y + z\})$ has variety $\{\langle a, -a, a \rangle \ : \ a \in \mathbb{C}\}$.

The ideal $\mathrm{I}(\{x(x-1), y + x - 3, z - x\})$ has variety $\{\langle 0, 3, 0 \rangle, \langle 1, 2, 1 \rangle\}$.

There are two standard ways to combine ideals to obtain new ideals:

Definition 5 The **sum** of two ideals I and J, denoted by $I \oplus J$, is defined by $I \oplus J = \{f + g \ : \ f \in I, g \in J\}$. The **product** of two ideals I and J, denoted by $I \otimes J$, is defined to be the smallest ideal containing $\{fg \ : \ f \in I, g \in J\}$.

These two standard ways of combining two ideals have a straightforward effect on the corresponding varieties, as the next result indicates.

Lemma 1 (Hoong [14]) *For any pair of ideals I and J, we have:*

1. $\mathrm{V}(I \oplus J) = \mathrm{V}(I) \cap \mathrm{V}(J)$;
2. $\mathrm{V}(I \otimes J) = \mathrm{V}(I) \cup \mathrm{V}(J)$.

Furthermore, given two finite systems of polynomials, they can be combined in straightforward ways to obtain *generating sets* for the sum and product of their corresponding ideals, as the next result indicates.

Lemma 2 (Hoong [14]) *For any pair of ideals $I = \mathrm{I}(\{p_1, \ldots, p_m\})$ and $J = \mathrm{I}(\{q_1, \ldots, q_l\})$, we have:*

1. $I \oplus J = \mathrm{I}(\{p_1, \ldots, p_m, q_1, \ldots, q_l\})$;
2. $I \otimes J = \mathrm{I}\left(\left\{p_i q_j \ : \ i \in \{1, \ldots, m\}, \ j \in \{1, \ldots, l\}\right\}\right)$.

Combining Lemmas 1 and 2, if we have two finite systems of polynomials, each with an associated set of solutions, then it is easy to obtain a new system of polynomials whose solutions are precisely the union or intersection of these two sets

of solutions. For example, in the case of the intersection, we simply concatenate the two systems of polynomials.

3 Representing constraints by systems of polynomials

In this section we show how various forms of constraints can be mapped to a system of polynomials whose set of solutions is precisely the set of combinations of values allowed by the constraint. If we do this for each constraint in a problem, then, by the observation at the end of Section 2, we can simply concatenate the systems of polynomials we obtain to create a system which represents the entire problem.

We shall assume for simplicity that each variable in the constraints we are trying to represent has a specified **domain** of possible values, and that these values are represented by natural numbers. Hence each constraint allows some set of tuples of natural numbers.

We shall represent a constraint over some subset of the variables $x_1, \ldots, x_n$ by a system of polynomials from $\mathbb{C}[x_1, \ldots, x_n]$. The solutions to this system will be precisely the assignments (of natural numbers) that satisfy the constraint.

First note that we can restrict the domain of possible values for each individual variable to a specified finite set by including in our system a polynomial of the following form for each variable.

Definition 6 The **domain polynomial** of a variable x with domain D is the polynomial $\prod_{j \in D}(x - j)$.

In this paper we shall assume that all the variables have *finite* domains and we will always include a domain polynomial for each variable in the system used to represent a constraint (even though they are sometimes redundant). This has the advantage[2] that it makes our solution methods complete, and provides a simple bound on the complexity (see Sections 4 and 5). Note that the system of polynomials consisting of just the domain polynomials for the variables $x_1, \ldots, x_r$ represents the rather trivial constraint which allows *all possible* combinations of values (from the domains) for these variables.

Example 2 (Constant constraints) Probably the simplest non-trivial form of constraint on a collection of variables $\langle x_1, \ldots, x_r \rangle$ is a **constant constraint** which allows only a single assignment, say $\langle \kappa_1, \ldots, \kappa_r \rangle$, assigning κ_i to x_i for $i = 1, \ldots, r$. This constraint is easily represented by the system of polynomials $\{x_1 - \kappa_1, x_2 - \kappa_2, \ldots, x_r - \kappa_r\}$, which has just one solution corresponding to the single allowed assignment.

Example 3 (Disjunctive constraints) Some forms of constraints, such as the propositional **clauses** used in the Satisfiability problem, are specified as a disjunction of simpler constraints. Using Lemmas 1 and 2, it follows that any such constraint can be represented by the system of polynomials consisting of the set of all products of all

[2] It has the further technical advantage that it ensures that the ideals generated by our systems of polynomials are *radical* [15, Lemma 8.19], but we will not use this property.

the polynomials in the systems representing the disjuncts. For example, consider the polynomial $\prod_{i\in\{1,\ldots,r\}}(x_i - \kappa_i)$. This polynomial is satisfied if, for some $i \in \{1, \ldots, r\}$, we have that x_i is assigned κ_i. In other words, at least one of the x_i's is assigned the associated κ_i. This type of polynomial can therefore be used to represent a clause in an instance of Satisfiability, by setting each κ_i to either 0 or 1, depending on whether the corresponding literal in the clause is negative or positive. For example, the clause $\neg x \vee y \vee \neg z$ is represented by the polynomial $x(y-1)z$, which is equal to $xyz - xz$.

Example 4 (Forbidding one assignment) We can generalize the previous example to any form of constraint which forbids precisely one assignment. Such constraints can always be represented by a single polynomial. For example, the constraint on a collection of variables $\langle x_1, \ldots, x_r \rangle$ each with domain $\{\kappa_1, \kappa_2, \ldots, \kappa_r\}$ that forbids just the assignment of κ_i to x_i for $i = 1, 2, \ldots, r$ can be represented by the polynomial

$$p(x_1, x_2, \ldots, x_r) = \prod_{j\neq 1}(x_1 - \kappa_j) \prod_{j\neq 2}(x_2 - \kappa_j) \cdots \prod_{j\neq r}(x_r - \kappa_j)$$

which allows all assignments except for the single forbidden assignment.

Example 5 (Linear equations) Any linear equation involving variables $x_1, \ldots, x_r$ can be written in the form $p(x_1, \ldots, x_r) = 0$, where p is a special form of polynomial in which $\sum \alpha_i \leq 1$ for all terms $\kappa {x_1}^{\alpha_1} \cdots {x_r}^{\alpha_r}$. This means that any constraint which is expressed by one or more linear equations is naturally represented by a system of polynomials. This includes many of the constraints encountered in Integer Programming problems. Moreover, the MiniZinc constraint system [16] can translate a wide variety of constraint problems into collections of linear constraints.

Example 6 (Inequality) Consider the constraint "$x \leq y + c$" over the variables x and y which each have the domain $\{1, 2, \ldots, d\}$. The set of solutions to this constraint can be represented by the system of polynomials $P = \{p_i(x, y) : i = 1, \ldots, d\}$, where

$$p_i(x, y) = \prod_{1\leq j\leq i+c} (x - j) \prod_{i+1\leq j\leq d} (y - j).$$

To see this, note that, for any $i \in \{1, \ldots, d\}$, any solution to $x \leq y + c$ will satisfy either $x \leq i + c$ or $y \geq i + 1$, and hence be a solution to $p_i(x, y) = 0$. Conversely, any assignment to x and y such that $x > y + c$ will fail to satisfy the equation $p_i(x, y) = 0$ where i is the value assigned to y. Hence this system of polynomials generates an ideal whose variety is exactly the set of solutions to the constraint.

Example 7 (Sum-Inequality) Consider the constraint "$w + x < y + z$" over the variables w, x, y and z which each have the domain $\{1, 2, \ldots, d\}$. By a similar argument to the one in Example 6, the set of solutions to this constraint can be represented by the system of polynomials $P = \{p_i(w, x, y, z) : i = 1, \ldots, 2d\}$, where

$$p_i(w, x, y, z) = \prod_{1\leq j\leq i-1} (w + x - j) \prod_{i+1\leq j\leq 2d} (y + z - j).$$

Example 8 (All-different) A very commonly-used constraint is the constraint that requires each of the variables in a given list to take a different value. This can be

expressed as a collection of disequalities, $x \neq y$, on each pair of variables x and y in the list. Since we are assuming that all the variables are restricted to integer values by the domain polynomials, we can express each of these disequalities using a disjunctions of inequalities, by imposing the constraint that $x \leq y - 1$ or $y \leq x - 1$. Hence we can use the methods described in Examples 3 and 6 to build a suitable system of polynomials to represent all-different constraints.

Example 9 (Table constraints) Any constraint which specifies an explicit list of allowed combinations can be seen as a disjunction of constant constraints, (or a conjunction of constraints that forbid a single assignment), so in principle we can use the methods described in Examples 2, 3 and 4 to build a suitable system of polynomials to represent such constraints. However, in practice this method has a major limitation, in that it may produce systems of huge size.

Fortunately, there is a more sophisticated algorithm which computes a system of polynomials with a specified finite set of solutions in polynomial time in the size of that set [17]. In other words, there is a known algorithm which can be used to find a system of polynomials to represent any constraint specified by an explicit list of allowed tuples in polynomial time.

The examples above illustrate how constraints specified in various ways can be mapped to systems of polynomials whose solutions are precisely the combinations of values allowed by the constraints. In particular, we have shown that any constraint that is specified by an explicit list of allowed tuples can be converted to such a set of polynomials in polynomial time (Example 9). In principle, any constraint over a finite domain can be converted to an explicit list of allowed tuples and hence converted to a system of polynomials in this way. Moreover, certain common forms of implicitly specified constraints, such as clauses and linear equations, can be converted to polynomials in a more succinct way (Examples 3 and 5).

In general the conversion of constraints to a representation as a system of polynomials will obscure some features of the constraints but reveal others (see the discussion in Examples 13 and 17 below). It may also be viewed as a "compilation" technique, that converts the initial presentation of the problem into a form that is amenable to automated processing to reveal additional features, as we will now discuss.

4 Gröbner bases

Once we have constructed a system of polynomials whose solutions are precisely the assignments that satisfy a given collection of constraints, there are a number of questions we may wish to ask, such as: Does this system of polynomials have a solution? Does x_3 take the value 7 in all solutions? In this section we examine ways in which a system of polynomials can be manipulated to obtain a new system with the same set of solutions which makes answering such questions easier.

Note that any two systems of polynomials which generate the same ideal have the same sets of solutions, since the variety of a system of polynomials is equal to the variety of the ideal generated by that system. It turns out that a system of polynomials is particularly convenient to work with if it is a *Gröbner Basis* for the ideal generated

by that system. To define this property, we first have to define an ordering for the monomials in our polynomials.

Definition 7 A **monomial ordering**, $\prec$, on a set of variables $\{x_1, \ldots, x_n\}$ is a total ordering on monomials over $\{x_1, \ldots, x_n\}$ which satisfies the conditions:

1. For all monomials μ_1, μ_2 and μ_3 we have $\mu_1 \prec \mu_2 \Rightarrow \mu_1 \times \mu_3 \prec \mu_2 \times \mu_3$;
2. Every non-empty set of monomials has a smallest element with respect to $\prec$.

Example 10 One important monomial ordering is the **lexicographic** ordering, $\prec_{\text{lex}}$, which is defined by

$$x_1{}^{\alpha_1} \cdots x_n{}^{\alpha_n} \prec_{\text{lex}} x_1{}^{\beta_1} \cdots x_n{}^{\beta_n} \iff \langle \alpha_1, \ldots, \alpha_n \rangle < \langle \beta_1, \ldots, \beta_n \rangle ,$$

where the ordering on tuples of natural numbers shown on the right-hand side is the usual lexicographic ordering.

Example 11 Another important monomial ordering is the **total degree lexicographic** ordering, $\prec_{\text{d-lex}}$, which is defined by

$$x_1{}^{\alpha_1} \cdots x_n{}^{\alpha_n} \prec_{\text{d-lex}} x_1{}^{\beta_1} \cdots x_n{}^{\beta_n} \iff$$

$$\left(\sum \alpha_i < \sum \beta_i\right) \vee \left(\left(\sum \alpha_i = \sum \beta_i\right) \wedge \langle \alpha_1, \ldots, \alpha_n \rangle < \langle \beta_1, \ldots, \beta_n \rangle\right)$$

where the ordering on tuples of natural numbers is again the usual lexicographic ordering.

Once we have ordered the monomials, we can identify a distinguished term in every polynomial, in the following way.

Definition 8 Given a monomial ordering $\prec$ on the set of variables $\{x_1, \ldots, x_n\}$ and a non-zero polynomial $u = \sum_i \kappa_i \mu_i$ from $\mathbb{K}[x_1, \ldots, x_n]$, where each $\kappa_i \in \mathbb{K}$ and each μ_i is a monomial over $\{x_1, \ldots, x_n\}$, the **leading monomial** (LM) of u is the maximum of the μ_i under the monomial ordering $\prec$. If the leading monomial is μ_j (for some j), then the **leading term** (LT) of u is $\kappa_j \mu_j$.

Example 12 Consider the polynomial $4x^3 + 2y^2 + xy$ in the polynomial ring $\mathbb{C}[x, y]$. Under the lexicographic ordering, with $x \prec y$, the leading monomial is y^2 whilst the leading term is $2y^2$. Under total degree lexicographic ordering, with $x \prec y$, the leading monomial is x^3 whilst the leading term is $4x^3$.

Using the notion of leading term we can now define a form of division algorithm for multi-variate polynomials.

Definition 9 For any system of polynomials $P = \{p_1, \ldots, p_m\}$ from the ring $\mathbb{K}[x_1, \ldots, x_n]$, and any polynomial $u \in \mathbb{K}[x_1, \ldots, x_n]$, a **remainder** of u on division by P, denoted $u|_P$, is obtained by repeatedly performing the following "reduction" rule until it cannot be further applied:

Choose any $i \in \{1, \ldots, m\}$ such that $\text{LT}(p_i)$ divides some term τ of u and replace u with $u - \frac{\tau}{\text{LT}(p_i)} p_i$.

Unfortunately, in the general case the remainder of a given polynomial on division by a given system of polynomials is not uniquely defined. This is because the division algorithm as described in Definition 9 is non-deterministic: it can produce different results depending on the order in which the polynomials in P are considered. Furthermore, in general, even when a polynomial f is in the ideal $\mathrm{I}(P)$, it is not always the case that $f|_P = 0$. For example, x cannot be reduced by either polynomial in the system $\{x^2 + x, x^2\}$, although it can clearly be written as the first polynomial minus the second, and hence belongs to the ideal generated by this system.

Definition 10 [11, 13, 15] A system of polynomials G from $\mathbb{K}[x_1, \ldots, x_n]$ is called a **Gröbner Basis** (with respect to a given monomial ordering) if it satisfies the property that $u|_G$ is uniquely defined for any $u \in \mathbb{K}[x_1, \ldots, x_n]$.

One very useful property of a Gröbner Basis is that it can be used to determine whether a given polynomial belongs to the ideal that the basis generates, as the next result shows.

Theorem 1 [11, 13, 14] *If G is a Gröbner Basis, then $f|_G = 0$ if and only if $f \in \mathrm{I}(G)$.*

The next example demonstrates the use of a Gröbner Basis to provide a convenient representation for a table constraint.

Example 13 The following relation over the set $D = \{1, 2, 3, 4\}$ appeared in [18, Example 13] where it was used as an example of a relation having a large amount of symmetry:

$$R = \{\langle 1,2,2,3,4\rangle, \langle 1,2,2,4,3\rangle, \langle 1,2,3,1,2\rangle,$$
$$\langle 2,4,2,3,4\rangle, \langle 2,4,2,4,3\rangle, \langle 2,4,3,1,2\rangle\}.$$

If we consider this relation as a constraint on the variables $\langle a, b, c, d, e\rangle$, then it can be represented by the following system of polynomials:

$$G = \{e^3 - 9e^2 + 26e - 24,\ b^2 - 6b + 8,$$
$$d + 2e^2 - 13e + 17,\ 2c - e^2 + 7e - 16,\ 2a - b\}.$$

This system of polynomials is a Gröbner Basis (with respect to the lexicographic monomial order where $e \prec b \prec d \prec c \prec a$). The two uni-variate polynomials in the set G correspond to unary constraints: they indicate that the second component of any solution satisfying the constraint has to be in the set $\{2, 4\}$ and the last component has to be in the set $\{2, 3, 4\}$. The remaining polynomials in the set G are bi-variate: they indicate relationships which were implicit in R and which are revealed by the Gröbner Basis. The fact that these polynomials are *linear* in the variables d, c and a indicates that for any $\langle b, e\rangle \in \{2, 4\} \times \{2, 3, 4\}$ there is *exactly one* $\langle a, b, c, d, e\rangle \in R$ and vice versa. In particular, if $\langle a, b, c, d, e\rangle \in R$ then $d = -2e^2 + 13e - 17$, $c = (e^2 - 7e + 16)/2$, and $a = b/2$.

One remarkable result established by Buchberger is that given any finite system of polynomials, and any fixed monomial ordering, it is possible to compute a new system of polynomials which generates the same ideal and is a Gröbner Basis [13, 14]. One

way to do this is to use Algorithm 1, which is known as Buchberger's Algorithm [13, 14].[3] The algorithm makes use of a construct called the S-*polynomial* of two given polynomials, which is defined as follows.

Definition 11 Given two monomials $x_1{}^{\alpha_1} \cdots x_n{}^{\alpha_n}$ and $x_1{}^{\beta_1} \cdots x_n{}^{\beta_n}$, their **lowest common multiple** (LCM) is $x_1{}^{\max(\alpha_1,\beta_1)} \cdots x_n{}^{\max(\alpha_n,\beta_n)}$.

Given polynomials u_1 and u_2 and a monomial ordering, the ***S*-polynomial** of u_1 and u_2, denoted S-Pol (u_1, u_2), is

$$\frac{\text{LCM}(\text{LM}(u_1), \text{LM}(u_2))}{\text{LT}(u_1)} u_1 - \frac{\text{LCM}(\text{LM}(u_1), \text{LM}(u_2))}{\text{LT}(u_2)} u_2 \ .$$

Example 14 Let $p = x^3 + z^3$ and $q = y^3 + z^3$, and take a total degree lexicographic ordering with $z \prec y \prec x$. Then we have S-Pol $(p, q) = y^3 z^3 - x^3 z^3$.

Note that although the leading terms of the two polynomials are eliminated in the calculation of their S-polynomial, the result may have a leading monomial which is higher in the monomial ordering than either of these original leading terms, as this example shows.

Algorithm 1 Buchberger's algorithm

```
GIVEN: P: Finite set of initial polynomials
RETURNS: G: Gröbner Basis for I(P)
BEGIN: G := P, T := {{g1, g2} : {g1, g2} ⊆ G}
while T ≠ ∅ do
    Select {g1, g2} from T and set T := T \ {{g1, g2}}
    h := S-Pol(g1, g2)
    h0 := h|G
    if h0 ≠ 0 then
        T := T ∪ {{g, h0} : g ∈ G}
        G := G ∪ {h0}
    end if
end while
```

Example 15 Consider a set Q of domain polynomials corresponding to distinct variables. Note that each domain polynomial in Q is uni-variate, and the leading terms of any two domain polynomials for distinct variables do not share a variable. This implies that the S-polynomial of any two domain polynomials in Q is reduced by those two polynomials to zero (this is Buchberger's first criterion, see Lemma 3.3.1 of [19]). Hence, applying Buchberger's Algorithm to Q will not change the set. It follows that any set of domain polynomials corresponding to any collection of distinct variables is a Gröbner Basis with respect to any monomial ordering.

[3] More sophisticated and efficient refinements of this algorithm have been developed, and are generally used in practice, but this simple version is sufficient to illustrate the technique and allow a straightforward complexity analysis.

Example 16 Recall from Example 6 that the constraint "$x \leq y$" over the variables x and y which each have the domain $\{1, 2, \ldots, d\}$ can be represented by the system of polynomials $P = \{p_i(x, y) : i = 1, \ldots, d\}$, where

$$p_i(x, y) = \prod_{1 \leq j \leq i} (x - j) \prod_{i+1 \leq j \leq d} (y - j).$$

Whatever monomial ordering we choose, P is a Gröbner Basis. To see this, it is sufficient to show that S-Pol $(p_a, p_b)\,|_P = 0$ for all $a, b \in \{1, \ldots, d\}$. Without loss of generality assume $a < b$. Then the greatest common divisor of p_a and p_b is:

$$\prod_{1 \leq j \leq a} (x - j) \prod_{b+1 \leq j \leq d} (y - j)$$

After dividing both polynomials by this, the first will become a polynomial only in y, and the second a polynomial only in x. The result then follows by Lemma 3.3.1 of [19].

Once we have computed a Gröbner Basis, it is then much easier to obtain the solutions to the original set of polynomials (or determine that there are no solutions). In particular, we can decide whether the system has a solution using the following well-known result (for proof see, for example, [14])

Theorem 2 (Hilbert's Nullstellensatz) *Let* $\mathbb{K}$ *be an algebraically closed field and* I *be an ideal in* $\mathbb{K}[x_1, \ldots, x_n]$. *The variety* $\mathrm{V}(I)$ *is empty if and only if* I *contains the polynomial* 1.

Buchberger [11] used this result to show that it is possible to decide whether a system of polynomials has any solutions by simply calculating a Gröbner Basis (with respect to any monomial ordering) and then checking whether it contains a polynomial that is a non-zero constant. If so, the system has no solutions; otherwise, at least one solution exists. In our application the domain polynomials ensure that this solution gives each variable a value within its allowed finite domain.

Example 17 A collection of benchmark instances of the SATISFIABILITY problem are provided at www.satlib.org. These can easily be converted to systems of polynomials and then passed to a computer algebra package, such as Mathematica, to obtain a Groebner Basis.

Some of these benchmark instances contain random 3-clauses. For example, the instance "`uf20-01`" contains 91 random 3-clauses over 20 variables, as follows:

$$\{x_4 \vee \neg x_{18} \vee x_{19},\ x_3 \vee \neg x_5 \vee x_{18},\ \neg x_5 \vee \neg x_8 \vee \neg x_{15},$$
$$x_7 \vee \neg x_{16} \vee \neg x_{20},\ \neg x_7 \vee x_{10} \vee \neg x_{13},\ \neg x_9 \vee \neg x_{12} \vee x_{17},\ \cdots,$$
$$x_9 \vee x_{17} \vee \neg x_{19},\ \neg x_2 \vee x_{12} \vee x_{17},\ x_4 \vee \neg x_5 \vee \neg x_{16}\}.$$

A straightforward mapping to polynomials as described in Example 3 gives the following system of 91 polynomials:

$$\{(1-x_4)\,x_{18}\,(1-x_{19})\,,\ (1-x_3)\,x_5\,(1-x_{18})\,,\ x_5x_8x_{15},$$
$$(1-x_7)\,x_{16}x_{20},\ x_7\,(1-x_{10})\,x_{13},\ x_9x_{12}\,(1-x_{17})\,,\ \cdots,$$
$$(1-x_9)\,(1-x_{17})\,x_{19},\ x_2\,(1-x_{12})\,(1-x_{17})\,,\ (1-x_4)\,x_5x_{16}\}.$$

If we combine this system with domain polynomials $x_i(x_i-1)$, restricting each variable x_i to the Boolean values 0 or 1, then the built-in **GroebnerBasis** command of Mathematica calculates the following Gröbner Basis for the combined system in around 0.7 s:[4]

$$\{x_{20}-1, x_{19}^2-x_{19}, x_{18}-x_{19}, x_{17}-1, x_{16}, x_{15}-1, x_{14}-1, x_{13}x_{19},$$
$$x_{13}^2-x_{13}, x_{12}, x_{11}-x_{19}, x_{10}x_{19}-x_{19}, x_{13}x_{10}-x_{10}+x_{19}, x_{10}^2-x_{10}, x_9x_{19}-x_{19},$$
$$x_{13}x_9-x_9-x_{13}+1, x_9x_{10}-x_{19}, x_9^2-x_9, x_8x_{19}-x_{19}, x_{13}x_8-x_8+x_{19}, x_8x_{10}-x_8,$$
$$x_8x_9-x_{19}, x_8^2-x_8, x_7, x_6x_{19}, x_{13}x_6-x_6-x_{13}-x_{19}+1, x_{10}x_6-x_6-x_{10}+1,$$
$$x_6x_9-x_9+x_{19}, x_6x_8, x_6^2-x_6, x_5, x_4x_{19}-x_{19}, x_{13}x_4-x_4+x_{19}, x_4x_{10}-x_{10},$$
$$x_4x_9-x_{19}, x_4x_8-x_8, x_6x_4-x_4-x_6+1, x_4^2-x_4, x_3-x_{19}, x_2-x_{19}, x_1+x_{19}-1\}.$$

By Theorem 2, this indicates that this instance is soluble. Furthermore, many of the polynomials in this basis are linear and univariate, indicating that the corresponding variables have a fixed value in all solutions. For example, the polynomial $x_{20}-1$ reveals that the variable x_{20} must be assigned the value 1 in all solutions. Other features can also be immediately read off from this set of polynomials, that are not obvious from the original clauses, such as the fact that variables x_{18} and x_{19} must take the same value in all solutions.

Once we have determined that a system has a solution, we can build such a solution by constructing progressively larger satisfying partial assignments which we can verify will extend to a complete solution. One way to do this with Gröbner Bases is to use the Extension Theorem [14, p. 115], which requires special monomial orderings (usually the lexicographic monomial ordering is used). However, because of the domain polynomials we only need to consider a fixed finite number of possible values for each new variable included in the partial assignment. This means we can use simple techniques to iteratively generate and test possible solutions.

One way to build a complete solution in this way involves generating a Gröbner Basis G for the system just once. We then use this to check whether any particular partial assignment s to the constraint problem extends to a solution. We can do this by checking whether the polynomial p_s which forbids precisely that assignment (Example 4) is in the ideal generated by G, and we can decide this by computing $p_s|_G$. If $p_s|_G = 0$, then p_s is in the ideal, and hence the assignment s cannot be part of a solution to the constraint problem, otherwise p_s is not in the ideal, and hence s can be extended to a solution.

[4] All timings reported in this paper were obtained using Mathematica 9 running under Windows 7 on an Intel Core i7-3770 CPU processor running at 3.4 GHz.

An alternative method is to add polynomials representing the partial assignment (recall Example 2) to the system and regenerate the Gröbner Basis. We then use this to determine the satisfiability of the new system.

Using either of these methods we can construct solutions incrementally without backtracking. Hence a solution to a problem with n variables, each with d possible values, can be found with at most $n \times d$ iterations.

5 Complexity

For a general system of polynomials P, the worst-case complexity of Buchberger's Algorithm is doubly exponential in the size of P [15, p. 513], and hence it can only be used with confidence for very small systems. The complexity of computing a Gröbner Basis in general is discussed in [20] and in [14].

However, the systems of polynomials we are considering have the special property that they contain a domain polynomial for each variable, limiting the possible values for that variable to a fixed finite set. The ideal generated by a system of polynomials with only a finite number of solutions is said to be *zero-dimensional*. Our next result, Lemma 3 shows that the presence of domain polynomials ensures that Buchberger's algorithm terminates in singly exponential time. This conclusion also follows from the existing result that, for any zero-dimensional ideal, a Gröbner Basis with respect to certain monomial orderings can be obtained in singly exponential time [21]. However, Lemma 3 differs from this previous result by using the stronger condition of the existence of domain polynomials to produce a much simpler proof, and to remove any limitation on the monomial orderings which can be used.

Lemma 3 *Let P be a system of polynomials over $x_1, \ldots, x_n$ which contains a domain polynomial for each variable of degree at most d. The total number of new polynomials added to P by Buchberger's Algorithm is at most d^n, each of which contains at most d^n monomials.*

Proof Note that in Algorithm 1 each polynomial h is reduced by P before being added to the set G. If any term of h contains any variable x whose exponent is larger than d, then it will be reduced by the domain polynomial to give a lower power of x. Moreover, each new polynomial added to G must have a different leading monomial from all current members of G, or else it will be further reduced before being added. The number of distinct monomials over $x_1, \ldots, x_n$, where each power is at most $d - 1$ is d^n. Further, each of these polynomials has at most d^n monomials with this property which precede it in any monomial ordering, so each polynomial added contains at most d^n monomials. □

Lemma 3 shows that the number of times that we can generate a new non-zero polynomial (line 8 of Algorithm 1) is bounded by d^n. Each of these adds at most $|P| + d^n$ elements to T (line 9). Therefore, the central loop (lines 4–12) executes at most $|P|^2 + (|P| + d^n)d^n < (|P| + d^n)^2$ times. Calculating the S-polynomials and reduced polynomials (lines 6 and 7) are both polynomial in the *size* of their input (also polynomial-size in $|P| + d^n$). Therefore, Algorithm 1 runs in polynomial time in $|P| + d^n$.

Lemma 3 shows that the worst-case complexity of computing a Gröbner Basis representing a finite-domain constraint problem is singly exponential, and hence has the same asymptotic order of complexity as other general solution methods, such as backtrack search.

On the other hand, our next example demonstrates that a Gröbner Basis representing a constraint problem can be very large, even for a simple problem. First we need a technical lemma.

Lemma 4 *For any positive integers n and d, if p is a non-zero polynomial over the variables* $x_1, \ldots, x_n$ *that allows every integer assignment with* $1 \leq x_1 \leq x_2 \cdots \leq x_n \leq d$, *then the leading term of p under any monomial ordering has total degree at least d.*

Proof We prove the result for any fixed monomial ordering by induction on n and d. When $n = 1$ the result follows from the fundamental theorem of algebra, since any non-zero polynomial in one variable of degree $d - 1$ can have at most $d - 1$ roots.

When $d = 1$ the result holds trivially, since any non-zero polynomial with a root must have degree at least one.

Now choose any $n > 1$ and any $d > 1$, and assume that the result holds for all smaller values of n, and all smaller values of d with that n. Let p be a polynomial over the variables $x_1, \ldots, x_n$ that allows every integer assignment with $1 \leq x_1 \leq x_2 \cdots \leq x_n \leq d$.

Dividing by $(x_n - d)$, we can express p as $(x_n - d)q + r$ for some polynomial q over $x_1, \ldots, x_n$ and some polynomial r over $x_1, \ldots, x_{n-1}$.

Since r must allow all integer assignments with $1 \leq x_1 \leq x_2 \cdots \leq x_{n-1} \leq d$, we know by our assumption that if it is non-zero then its leading term has total degree at least d.

Similarly, since q must allow all integer assignments with $1 \leq x_1 \leq x_2 \cdots \leq x_n \leq d - 1$, we know by our assumption that if it is non-zero then its leading term has total degree at least $d - 1$. Hence, if q is non-zero the leading term of $(x_n - d)q$ has total degree at least d, and in all cases the leading term of $(x_n - d)q + r$ has total degree at least d, which gives the result. □

Example 18 Consider a constraint problem with variables $x_1, \ldots, x_n$, each with domain $\{1, \ldots, d\}$, and overlapping inequality constraints specifying that $x_1 \leq x_2 \leq \cdots \leq x_n$.

We can construct a system of polynomials, P, whose solutions are the solutions to this constraint problem simply by combining the Gröbner Bases for the inequality constraint on each pair of adjacent variables, as given in Example 16. However, the resulting system of $d(n - 1)$ polynomials is not a Gröbner Basis for the ideal, I, generated by P, as we shall now demonstrate.

First, we know from Lemma 4 that with any monomial ordering, the leading term of any polynomial in I has total degree at least d.

For any sequence $\sigma = (i_1, i_2, \ldots, i_d)$ such that $1 \leq i_1 \leq i_2 \leq \cdots \leq i_d \leq n$, we define the polynomial p_σ as follows: $p_\sigma(x_{i_1}, \ldots, x_{i_d}) = (x_{i_1} - 1)(x_{i_2} - 2) \cdots (x_{i_d} - d)$. Note that any assignment of values in the set $\{1, \ldots, d\}$ to the variables of p_σ that does not satisfy $p_\sigma = 0$ must assign $x_{i_1} > 1$ and $x_{i_d} < d$, so there must be an index j where $x_{i_j} > j$ and $x_{i_{j+1}} < j + 1$. But this contradicts the constraints, which require that any

solution is non-decreasing. Hence any solution to the constraint problem we are considering is a solution to $p_\sigma = 0$, so $p_\sigma \in I$ for all possible choices of σ.

With any monomial ordering, the leading monomials of the polynomials p_σ include all possible monomials of total degree d, and hence generate the leading terms of all polynomials in I. It follows that these polynomials form a Gröbner Basis for I with respect to any monomial ordering [14, p. 74]. Moreover, this Gröbner Basis is reduced, and hence unique [14, p. 90].

There are $\binom{n+d-1}{d}$ distinct ways to choose the sequence σ, and hence $\binom{n+d-1}{d}$ distinct polynomials of the form p_σ. For example, when $n = 10$ and $d = 5$ this set of polynomials contains $\binom{14}{9} = 2002$ polynomials. It takes around 90 s to calculate this Gröbner Basis using Mathematica.

Example 18 shows that the size of the Gröbner Basis grows rapidly with n and d in this case, whatever monomial ordering is used, and can be expensive to compute, even though the underlying constraint problem is very straightforward. This is because the Gröbner Basis encodes information about *all* of the relationships that can be deduced between all of the variables in the problem. In the next section we consider modifying Buchberger's Algorithm to obtain a system of polynomials which encodes only *local* relationships among the variables.

6 A weaker form of basis

In order to obtain a polynomial-time variant of Buchberger's Algorithm, we will now describe a modified form of the basic algorithm which will generally be incomplete, in the sense that it generates a system of polynomials which may not always be a Gröbner Basis, but can still provide useful information, as we will establish below. This approach has been previously studied by Clegg et al. [6] for the special case of constraint problems over Boolean domains (including all instances of the SATISFIABILITY problem). In this section we will extend their approach to a more general setting by defining a new form of basis for a polynomial system.

We first define a link between Gröbner Bases and a certain *proof system*.

Definition 12 Given a system of polynomials P from $\mathbb{K}[x_1, \ldots, x_n]$ we define a **derivation-proof** from P of a polynomial f to be a sequence of polynomials $\langle f_1, \ldots, f_b \rangle$ such that $f_b = f$, and for $i = 1, \ldots, b$, we have:

$$\exists p \in P \cup \{f_1, \ldots, f_{i-1}\}, \exists u \in \mathbb{K}[x_1, \ldots, x_n],\ f_i = pu \quad \vee$$
$$\exists f_j, f_k \in \{f_1, \ldots, f_{i-1}\},\ f_i = f_j + f_k$$

If there exists a derivation-proof of f from P, then we denote this by $P \vdash f$.

Notice that a polynomial f has a derivation-proof from some system of polynomials P if and only if f is in the ideal generated by P. Hence, by Theorem 1, given a Gröbner Basis G, and a polynomial f, $f|_G = 0$ if and only if $G \vdash f$.

In order to obtain more efficient algorithms for processing polynomials we will now restrict the notion of a derivation-proof, in order to define a weaker form of proof system, and a correspondingly weaker notion of basis. We do this by restricting the polynomials that can be used in a derivation.

Definition 13 A derivation-proof $\langle f_1, \ldots, f_b \rangle$ is called a ϕ**-proof** if the polynomial derived at every step satisfies a specified property ϕ (that is, $\phi(f_i)$ holds, for $i = 1, \ldots, b$). If there exists a ϕ-proof of f from P then we write $P \vdash_\phi f$.

We will say that a system of polynomials G is a ϕ**-basis** if, for any polynomial f satisfying ϕ, $f|_G = 0$ if and only if $G \vdash_\phi f$.

Our next theorem will establish a sufficient condition for a system of polynomials to be a ϕ-basis, for a broad class of properties ϕ. The proof is similar in style to the usual proof that "Buchberger's S-pair criterion" is a sufficient condition to establish a Gröbner Basis (see, for example, [14, p. 83, Theorem 6]). First we need some technical definitions, and a standard lemma.

Definition 14 Given a system of polynomials P, and a property ϕ, we say that a polynomial f has a ϕ**-representation** over P if f can be written as $\sum_i p_i h_i$ where each $p_i \in P$, and each $p_i h_i$ satisfies ϕ.

Given a system of polynomials P, and a monomial ordering $\prec$, we say that a polynomial f is **semi-reducible** over P with respect to $\prec$ if f can be written as $\sum_i p_i h_i$ where each $p_i \in P$, and each $p_i h_i$ satisfies $\text{LM}(p_i h_i) \preceq \text{LM}(f)$.

Lemma 5 ([14, p. 81, Lemma 5]) *Suppose we have a linear sum of polynomials* $\sum_{i=1}^{s} c_i f_i$*, where* $c_i \in \mathbb{K}$ *and* $\text{LM}(f_i) = \mu$ *for all i.*

If $\text{LM}(\sum_{i=1}^{s} c_i f_i) \prec \mu$*, then* $\sum_{i=1}^{s} c_i f_i$ *is a linear sum, with coefficents in* $\mathbb{K}$*, of the S-polynomials* $\text{S-Pol}\left(f_j, f_k\right)$ *for* $1 \leq j, k \leq s$*. Furthermore,* $\text{LM}(\text{S-Pol}\left(f_j, f_k\right)) \prec \mu$ *for each j, k.*

Theorem 3 *Let* $\prec$ *be a monomial ordering, and let* ϕ *be a property of polynomials such that if* $\phi(f)$ *holds, and* $\text{LM}(g) \preceq \text{LM}(f)$*, then* $\phi(g)$ *also holds.*

Let Q be a system of polynomials such that, for all $q_1, q_2 \in Q$*, if* $\text{S-Pol}(q_1, q_2)$ *satisfies* ϕ*, then* $\text{S-Pol}(q_1, q_2)$ *is semi-reducible over Q with respect to* $\prec$*.*

Then, for any polynomial p satisfying ϕ*, the following are equivalent:*

1. $Q \vdash_\phi p$;
2. *p is* ϕ*-representable over Q;*
3. *p is semi-reducible over Q with respect to* $\prec$*;*
4. $p|_Q = 0$

Proof We first show that (2) $\Leftrightarrow$ (3).

To establish that (2) $\Rightarrow$ (3), assume for contradiction that there exists some polynomial p which is ϕ-representable over Q but not semi-reducible over Q with respect to $\prec$. Since p is ϕ-representable, we know that p can be expressed as $\sum h_i q_i$, where each $q_i \in Q$ and each $h_i q_i$ satisfies ϕ. Since p is not semi-reducible, we know that in any such representation $\text{LM}(p) \prec \max_i\{\text{LM}(h_i q_i)\}$. Choose a representation for which $\max_i\{\text{LM}(h_i q_i)\}$ is minimal (according to $\prec$) and call this monomial μ_p.

If we set $I = \{i : \text{LM}(h_i q_i) = \mu_p\}$, then we obtain:

$$p = \sum_{i \in I} \text{LT}(h_i) q_i + \sum_{i \in I} (h_i - \text{LT}(h_i)) q_i + \sum_{i \notin I} h_i q_i$$

where all of the polynomials in the first summation have leading monomial μ_p, and all polynomials in the other summations have a leading monomial which is strictly smaller (according to $\prec$).

By Lemma 5, we can express the first summation as a linear sum of S-polynomials of pairs of $\text{LT}(h_i)q_i$ terms. We can then rewrite these S-polynomials, as follows, where we write L_{jk} for $\text{LCM}(\text{LT}(q_j), \text{LT}(q_k))$:

$$\begin{aligned}
\text{S-Pol}\left(\text{LT}(h_j)q_j, \text{LT}(h_k)q_k\right) &= \frac{\mu_p}{\text{LT}(h_j)\,\text{LT}(q_j)}\ \text{LT}(h_j)q_j - \frac{\mu_p}{\text{LT}(h_k)\,\text{LT}(q_k)}\ \text{LT}(h_k)q_k \\
&= \frac{\mu_p}{\text{LT}(q_j)}\ q_j - \frac{\mu_p}{\text{LT}(q_k)}\ q_k \\
&= \frac{\mu_p}{L_{jk}}\left(\frac{L_{jk}}{\text{LT}(q_j)}\ q_j - \frac{L_{jk}}{\text{LT}(q_k)}\ q_k\right) \\
&= \frac{\mu_p}{L_{jk}}\ \text{S-Pol}\left(q_j, q_k\right).
\end{aligned}$$

By Lemma 5, each $\text{S-Pol}\left(\text{LT}(h_j)q_j, \text{LT}(h_k)q_k\right)$ has leading monomial strictly smaller than μ_p, and hence so does each $\text{S-Pol}\left(q_j, q_k\right)$. By our choice of property ϕ, this means that each $\text{S-Pol}\left(q_j, q_k\right)$ satisfies ϕ. By our assumption about Q, this mean that each $\text{S-Pol}\left(q_j, q_k\right)$ is semi-reducible over Q with respect to $\prec$. Hence we can express each $\text{S-Pol}\left(\text{LT}(h_j)q_j, \text{LT}(h_k)q_k\right)$ as a sum of polynomials from Q with polynomial coefficients where each term in the summation has leading monomial strictly smaller than μ_p. This contradicts the minimality of μ_p and hence establishes the result that (2) $\Rightarrow$ (3).

The converse is straightforward: by the choice of property ϕ, it follows immediately from Definition 14 that any polynomial which is semi-reducible and satisfies ϕ is also ϕ-representable. Hence (3) $\Rightarrow$ (2).

Next we show that (1) $\Leftrightarrow$ (2). Let p be any polynomial such that $Q \vdash_\phi p$. We will show that p is ϕ-representable over Q by induction on the length of the ϕ-proof of p from Q (see Definition 12). For the base case, if $p = qu$, for some $q \in Q$, and p satisfies ϕ, then p is clearly ϕ-representable over Q.

Now assume that p has a ϕ-proof of length i, and the result holds for all shorter proofs.

If $p = f_j u$ for some f_j with a shorter proof, then we know by the induction hypothesis that f_j is ϕ-representable over Q, and hence semi-reducible over Q with respect to $\prec$, by the argument above. This means that $f_j = \sum h_i q_i$ where $\text{LM}(h_i q_i) \preceq \text{LM}(f_j)$. Hence $p = \sum h_i q_i u$ where $\text{LM}(h_i q_i u) \preceq \text{LM}(f_j u) = \text{LM}(p)$. Hence in this case p is semi-reducible over Q with respect to $\prec$, and hence ϕ-representable, by the argument above.

If $p = f_j + f_k$ for some f_j, f_k with shorter proofs, then we know by the induction hypothesis that f_j and f_k are ϕ-representable over Q. Adding these representations gives a ϕ-representation for p over Q.

Hence, in all cases p is ϕ-representable, so we have shown that (1) $\Rightarrow$ (2).

The converse is immediate from Definitions 12 and 14, since any polynomial which is ϕ-representable over a set Q clearly has a ϕ-proof from Q.

Finally, we show that (3) $\Leftrightarrow$ (4). Assume for contradiction that there exists a polynomial p satisfying ϕ which is semi-reducible over Q with respect to $\prec$, but with $p|_Q \neq 0$. Chose such a polynomial whose leading monomial is as small as possible

(according to $\prec$). Since p is semi-reducible, we know that p can be expressed as $\sum h_i q_i$, where each q_i in Q and each $h_i q_i$ satisfies $\text{LM}(h_i q_i) \preceq \text{LM}(p)$. In this summation there must exist at least one $h_i q_i$ such that $\text{LM}(h_i q_i) = \text{LM}(p)$, and for this value of i we have $\text{LT}(q_i) | \text{LT}(p)$. Now consider the polynomial $p' = p - \frac{\text{LT}(p)}{\text{LT}(q_i)} q_i$.

By the results above, $Q \vdash_\phi p$. Hence $Q \vdash_\phi p'$, so p' is also semi-reducible over Q with respect to $\prec$, by the results above. Since the leading monomial of p' is strictly smaller than the leading monomial of p, we know by the choice of p that $p'|_Q = 0$. However this implies that $p|_Q = 0$, which contradicts the choice of p. Hence we have shown that (3) $\Rightarrow$ (4).

The converse follows immediately from Definition 9, since any polynomial which reduces to 0 over Q must be semi-reducible over Q. □

We have shown that the conditions described in Theorem 3 are sufficient to ensure that the system of polynomials Q is a ϕ-basis. Now consider a modified form of the original Buchberger Algorithm, as shown in Algorithm 1, which only considers S-polynomials which satisfy the property ϕ. In other words, the algorithm only performs lines 7–11 when the S-polynomial h satisfies ϕ. Such a modified form of the algorithm will be called a **ϕ-truncated** Buchberger Algorithm. When ϕ is a property of the form specified in Theorem 3, the system of polynomials, Q, computed by a ϕ-truncated Buchberger Algorithm will have the property that for any $q_1, q_2 \in Q$, if S-Pol(q_1, q_2) satisfies ϕ, then S-Pol(q_1, q_2) is semi-reducible over Q. Hence Q satisfies the conditions of Theorem 3, and hence will be a ϕ-basis.

Even though a ϕ-basis computed in this way is not guaranteed to be a Gröbner Basis, it does generate the same ideal as the original system of polynomials, and can reveal significant information about this ideal, and hence the corresponding set of solutions. Moreover, we will show below that for some classes of problems computing a ϕ-basis in this way (for an appropriate choice of ϕ) is sufficient to decide whether any solutions exist.

Moreover, for some choices of the property ϕ, the time complexity of the ϕ-truncated Buchberger Algorithm is much lower than the time complexity of the complete Buchberger Algorithm.

The first specific property ϕ that we consider is based on the **total degree** of the monomials occurring in a polynomial. We will say that a polynomial satisfies the property [degree $\leq m$] if, for each of its monomials ${x_1}^{\alpha_1} \cdots {x_n}^{\alpha_n}$, the sum of the exponents $\sum_{i=1}^{n} \alpha_i$ is at most m. We first need to ensure that the monomial ordering we are using has the property that if μ_1 satisfies the [degree $\leq m$] property, and $\mu_2 \prec \mu_1$, then μ_2 also satisfies the [degree $\leq m$] property. Such an ordering is called a **graded** monomial ordering. Many such orderings exist, including the standard total degree lexicographic ordering [14, 15] (see Example 11).

Lemma 6 *Let P be a system of polynomials from $\mathbb{K}[x_1, \ldots, x_n]$. The total number of new polynomials added to P by the* [degree $\leq m$]*-truncated Buchberger Algorithm using any graded monomial ordering is at most $\binom{n+m}{m}$, each of which contains fewer than $\binom{n+m}{m}$, monomials.*

Proof Similar to the proof of Lemma 3, except that the number of distinct monomials over $x_1, \ldots, x_n$ satisfying [degree $\leq m$] is $\binom{n+m}{m}$, and each of these therefore has fewer than $\binom{n+m}{m}$, monomials which precede it in any graded monomial ordering. □

By a similar argument to the argument given for the full algorithm above, this result implies that the truncated algorithm is polynomial-time in $|P|$ and $\binom{n+m}{m}$. This is $O(n^m)$, and therefore polynomial in n for fixed m.

7 Local consistency

In this section we show that the local consistency algorithms used in constraint programming can be expressed by ϕ-proofs on the corresponding systems of polynomials, for a suitable choice of property ϕ. By relating this property to the property [degree $\leq m$] considered in the previous section, we show that a local consistency algorithm can be simulated by a truncated Buchberger Algorithm, and hence any information obtained by enforcing local consistency can be obtained by performing a truncated Buchberger Algorithm.

A constraint problem is said to be **strong k-consistent** if any consistent assignment to $k-1$ or fewer variables can be extended to a consistent assignment to any additional variable. Strong k-consistency can be enforced in polynomial-time (for any fixed k) [22]. One method of achieving this is given in Algorithm 2.

Algorithm 2 (Enforcing strong (k+1)-consistency)

1. For each set of k or fewer variables, define a new constraint, equal to the relational join of the projections of all constraints onto those variables.
2. Replace each constraint by the relational join of that constraint with the projections of all other constraints onto its variables.
3. Repeat (2) until no changes are made.

It is known that, for several broad classes of constraint problems, the existence of a solution can be decided by enforcing strong k-consistency. These include problems with bounded tree-width [23] and problems where the constraints are binary and *max-closed* [24].

We now observe that if our original constraints each involve at most k variables, then the operations described in Algorithm 2 only ever involve constraints on k or fewer variables. These operations can be simulated using operations on polynomials representing the constraints in such a way that the polynomials computed never involve more than k variables. (For example, projection of a constraint onto some subset of variables can be simulated using the Extension Theorem [14, p. 115] on some Gröbner Basis of the system which is generated using a lexicographic monomial ordering.)

We will say that a polynomial satisfies the property [#vars $\leq k$] if it involves at most k variables. Recall that if some system of polynomials over at most k variables has no solution then, by Hilbert's Nullstellensatz, any Gröbner Basis of this system must contain a constant polynomial. This implies that there is a [#vars $\leq k$]-proof of 1 from these polynomials.

Now consider any constraint problem represented by a system of polynomials P, where each polynomial in P satisfies [#vars $\leq k$]. Our observations above imply that, if applying strong $(k+1)$-consistency to this constraint problem results in an empty constraint, then there must be a [#vars $\leq k$]-proof of 1 from P.

Unfortunately, the property [#vars $\leq k$] does not satisfy the conditions of Theorem 3 with any standard monomial ordering, and it does not seem to be possible to

verify directly whether a polynomial has a [#vars $\leq k$]-proof of 1 using a [#vars $\leq k$]-truncated Buchberger Algorithm. However, the next two results show that the existence of a [#vars $\leq k$]-proof of 1 implies the existence of a [degree $\leq k \times (d-1)+1$]-proof of 1, where d is the maximum domain size.

Lemma 7 *Let Q be a set of domain polynomials for $x_1, \ldots, x_n$, with degree at most d. For any polynomial u from $\mathbb{K}[x_1, \ldots, x_n]$, if u satisfies* [#vars $\leq k$]*, then $u|_Q$ is uniquely defined, and satisfies both* [#vars $\leq k$] *and* [degree $\leq k \times (d-1)$].

Proof By Example 15, any set of domain polynomials forms a Gröbner Basis, and so we can reduce any polynomial by this set and obtain a unique result.

If a monomial in u is reduced by the domain polynomial for variable x, it must have already contained x and therefore the resulting monomials introduced cannot contain any more variables than the monomial being reduced. This means that reduction by the domain polynomials preserves the property [#vars $\leq k$].

After all possible reductions have been performed, no monomial can contain a power of any variable higher than $d-1$, else it could be further reduced. As no monomial can contain more than k variables, this means all monomials must satisfy [degree $\leq k \times (d-1)$]. □

Theorem 4 *Let $p_1, \ldots, p_m, u$ be polynomials from $\mathbb{K}[x_1, \ldots, x_n]$, and let Q be a set of domain polynomials for $x_1, \ldots, x_n$, of degree at most d. If each p_i satisfies* [#vars $\leq k$]*, and $\{p_1, \ldots, p_m\} \vdash_{[\#\text{vars} \leq k]} u$, then*

$$\{p_1|_Q, \ldots, p_m|_Q\} \cup Q \vdash_{[\text{degree} \leq k \times (d-1)+1]} u|_Q \ .$$

Proof We first note that $d \leq k \times (d-1)+1$, so each domain polynomial in Q satisfies [degree $\leq k \times (d-1)+1$]. Moreover, by Lemma 7, the polynomials $p_1|_Q, \ldots, p_m|_Q$ and $u|_Q$ also all satisfy [degree $\leq k \times (d-1)+1$].

Assume that the derivation-proof of u from $\{p_1, \ldots, p_m\}$ is given by the list of polynomials $\langle f_1, \ldots, f_b \rangle$. By induction on the length of this proof, b, we shall show that we can obtain a [degree $\leq k \times (d-1)+1$]-derivation of $u|_Q$ from $\{p_1|_Q, \ldots, p_m|_Q\} \cup Q$.

If f_b is derived by adding two earlier polynomials f_j and f_k, then it is straightforward to show that $f_b|_Q = f_j|_Q + f_k|_Q$, so by the induction hypothesis we are done.

If f_b is derived by multiplying some polynomial p from the set $\{p_1, \ldots, p_m\} \cup \{f_1, \ldots, f_{b-1}\}$ by some arbitrary polynomial h from $\mathbb{K}[x_1, \ldots, x_n]$, then, without loss of generality, we may assume that h is a polynomial consisting of a single variable, say x. There are then two cases to consider:

- $(px)|_Q = (p|_Q)x$; this means that $f_b|_Q = (p|_Q)x$, so by the induction hypothesis, we are done.
- $(px)|_Q \neq (p|_Q)x$; this can only happen if $(p|_Q)x$ contains some monomials where the exponent of x is equal to the degree of q_i, where q_i is the domain polynomial for x. In this case we can subtract appropriate multiples of q_i to obtain $(px)|_Q$, and hence, by the induction hypothesis, obtain a (possibly longer) [degree $\leq k \times (d-1)+1$]-proof of $f_i|_Q$ from $\{p_1|_Q, \ldots, p_m|_Q\} \cup Q$. □

It follows from Theorem 4 that we can check for the existence of a [#vars $\leq k$]-proof indirectly, using a [degree $\leq m$]-truncated Buchberger Algorithm, for $m = k \times (d-1)+1$. Hence, for a fixed domain size d, any constraint problem which can be decided by strong k-consistency can be decided in polynomial-time by a truncated Buchberger Algorithm.

8 Adaptive consistency

Another local consistency method which is commonly used in constraint programming is to vary the level of consistency which is enforced during the solution process depending on the local structure of the constraints. This technique is known as *adaptive consistency* [25], and is often implemented using a general algorithmic framework called *bucket elimination* [26].

The bucket elimination algorithm for a constraint problem proceeds as follows. It first orders the variables of the problem, and then partitions the constraints into separate collections, known as *buckets*. Each bucket is associated with a particular variable. The bucket associated with variable x contains all the constraints involving the variable x which do not involve any variables occurring higher in the ordering. In other words, each constraint is allocated to the bucket which is associated with whichever of the variables of that constraint occurs highest in the ordering. Buckets are then processed in order, according to the chosen variable ordering on their associated variables, from highest to lowest. When the bucket associated with a variable x is processed, an "elimination procedure" is performed over all the constraints in that bucket, yielding a new constraint that does not involve the variable x. This new constraint specifies the "effect" of the variable x on the remainder of the problem. In other words, it allows just those combinations of values that are allowed by the constraints in the bucket along with *some* value for x. The new constraint is then placed in the appropriate (lower) bucket and the processing continues until all buckets are processed, or some constraint is generated which allows no solutions, in which case the problem is declared to be insoluble.

If constraints are represented by systems of polynomials, in the way that we are proposing here, then bucket elimination can be implemented very simply using Gröbner Basis calculations. For each variable x, the bucket associated with x is assigned a subset of the polynomials in the given system, consisting of all polynomials over x (and possibly other variables lower in the ordering). If we calculate a Gröbner Basis for these polynomials, with respect to an appropriate *elimination ordering* [19], then the new constraint we require is represented by the subset of polynomials in this Gröbner Basis which do not involve x. These new polynomials can then be allocated to lower buckets and processing continues. An implementation of this algorithm as a Mathematica function is shown in Fig. 1, where the arguments are the set of polynomials, and the ordered list of variables (highest first). A subsidiary function is also shown in the figure, which associates each polynomial in the given set with the name of a variable indicating the bucket to which that polynomial is allocated. The time complexity of the bucket elimination algorithm is exponential in a structural parameter of the given constraint problem, called the *induced width* [26]. This value depends on the way in which the constraints overlap and the chosen variable ordering. For problems with (known) bounded tree-width it is possible to

```
bucketeliminate[polys_, vars_] := Module[{buckets={}, newpolys=polys},
  Scan[(buckets = Join[buckets, bucketassign[newpolys, vars]];
        newpolys = GroebnerBasis[Cases[buckets, {{#}, poly_} -> poly],
                                 vars, {#}];
        If[newpolys == {1},
          Print["Unsatisfiable at variable ", #]; Return[False]];) &, vars]]

bucketassign[polys_, vars_] :=
 Map[{Select[vars, Function[var, MemberQ[Variables[#], var]], 1], #}&, polys]
```

Fig. 1 Bucket elimination for a system of polynomials implemented in Mathematica

find (in polynomial time) a variable ordering such that the induced width is also bounded, so all such problems can be solved in polynomial time.

Example 19 Recall the simple constraint problem discussed in Example 18, with variables $x_1, \ldots, x_n$, each with domain $\{1, \ldots, d\}$, and inequality constraints specifying that: $x_1 \leq x_2 \leq \cdots \leq x_n$. It was shown in Example 18 that any Gröbner Basis for this problem contains $\binom{n+d-1}{d}$ polynomials, and the time required to compute such a Basis rises rapidly with n and d.

However, if the variables are ordered along the line, the induced width of this problem is 1, and the bucket elimination algorithm never has to deal with polynomials over more than 2 variables. This means that the time required to decide whether a solution exists increases only linearly with n.

For example, when $n = 10$ and $d = 5$ the Mathematica implementation shown in Fig. 1 is able to verify that the corresponding constraint problem is soluble in 0.015 s.

Example 20 Recall the "sum inequality" constraint defined in Example 7 and consider a simple constraint problem with variables $x_1, \ldots, x_n$, each with domain $\{1, \ldots, d\}$, and sum inequality constraints specifying that:

$$x_1 + x_2 < x_3 + x_4 < \cdots < x_{n-1} + x_n.$$

Problems of this form can be surprisingly challenging for standard constraint solvers, since no values can be eliminated from the domains of individual variables by consistency techniques. When $n = 4d$ such problems are unsatisfiable, but the time required to establish this using some conventional constraint solvers rises very rapidly with d [27]. In fact, the experiments reported in [27] show that when $d = 8$ and $n = 32$ the time required by the state-of-the-art solvers Minion and G12 on a naive model of this problem is already greater than 1200 s (20 min).

However, if the variables are ordered by subscript, the induced width of this problem is 3, and the bucket elimination algorithm never has to deal with polynomials over more than 4 variables. For example, when $d = 8$ and $n = 32$ the Mathematica implementation shown in Fig. 1 is able to verify that the corresponding constraint problem is insoluble in about 28 s.

9 Conclusion

We have shown how polynomials provide a powerful general language for expressing many different forms of constraint problems.

Once a problem has been expressed with a corresponding system of polynomials, we have described a standard technique to find a new system of polynomials with the same set of solutions, called a Gröbner Basis. A Gröbner Basis provides a convenient representation of all the solutions to a system of polynomials and hence of all solutions to the original constraint problem. It can be computed using a standard computer algebra package, and can be used to answer many different questions about the solutions in a straightforward way.

We have also shown that we may truncate the standard Gröbner Basis algorithm by defining a property which all polynomials added to the system must satisfy. If this property is suitably specified, then the algorithm generates a useful system of polynomials with the same solutions in polynomial time, albeit not generally a Gröbner Basis. We have demonstrated that we can use this truncated algorithm to achieve a kind of local consistency which can simulate the k-consistency algorithms commonly used in constraint programming.

Finally we have shown that adaptive consistency techniques for constraint problems can be implemented very easily using Gröbner Basis techniques.

This paper presents our initial findings on using Gröbner Basis techniques for solving constraint problems. Obvious directions for future research are to determine for which classes of constraint problems these techniques can efficiently find solutions, and to refine the techniques so that they can be implemented more efficiently.

We have emphasised throughout the paper that one advantage of the techniques we are proposing is that they allow constraint problems to be solved using standard mathematical software tools, such as the Gröbner Basis calculations provided by current computer algebra packages. More efficient, special-purpose, solvers could also be built to use, for example, the modified calculations described in Sections 6 and 7. It would also be interesting to adapt the recent techniques and tools for calculating a Gröbner Basis that are based on efficient linear algebra calculations [28, 29].

References

1. Rossi, F., van Beek, P., Walsh, T.(eds.): The Handbook of Constraint Programming. Elsevier (2006)
2. Aiba, A., Sakai, K., Sato, Y., Hawley, D., Hasegawa, R.: Constraint logic programming language CAL. In: Proceedings of the International Conference on Fifth Generation Computer Systems, pp. 263–76. Ohmsha Publishers (1988)
3. Granvilliers, L., Monfroy, E., Benhamou, F.: Symbolic-interval cooperation in constraint programming. In: Proceedings of ISSAC 2001, pp. 150–166 (2001)
4. Hong, H.: RISC-CLP(Real): constraint logic programming over real numbers. In: Benhamou, F., Colmerauer, A. (eds.) Constraint Logic Programming: Selected Research. MIT Press, Cambridge (1993)
5. Monfroy, E.: The constraint solver collaboration language of BALI. In: In Proceedings of JCSLP 1998, pp. 349–350 (1998)
6. Clegg, M., Edmonds, J., Impagliazzo, R.: Using the Groebner basis algorithm to find proofs of unsatisfiability. In: Proceedings of the 28th ACM Symposium on Theory of Computing, STOC '96, pp. 174–183. ACM Press (1996). doi:10.1145/237814.237860
7. Sakai, K., Sato, Y.: Boolean Gröbner bases. Tech. Rep. 488, ICOT (1988)
8. Sato, Y., Suzuki, A., Inoue, S., Nabeshima, K., Sakai, K.: Boolean Gröbner bases. J. Symb. Comput. **46**, 622–632 (2011). doi:10.1002/10.1016/j.jsc.2010.10.011
9. Conti, P., Traverso, C.: Buchberger algorithm and integer programming. In: AAECC, pp. 130–139 (1991)
10. Urbaniak, R., Weismantel, R., Ziegler, G.M.: A variant of the Buchberger algorithm for integer programming. SIAM J. Discrete Math. **10**(1), 96–108 (1997)

11. Buchberger, B.: Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen polynomideal (an algorithm for finding the basis elements in the residue class ring modulo a zero dimensional polynomial ideal). Ph.D. thesis, Mathematical Institute, University of Innsbruck, Austria (1965). (English translation in J. of Symbolic Computation, Special Issue on Logic, Mathematics, and Computer Science: Interactions, vol. 41, no. 3–4, pp. 475–511 (2006))
12. Buchberger, B.: Ein algorithmisches Kriterium fuer die Loesbarkeit eines algebraischen Gleichungssystems (an algorithmic criterion for the solvability of algebraic systems of equations). Aequationes Math. **3**, 374–383 (1970) (English translation: Buchberger, B., Winkler, F.: Groebner Bases and Applications. In: Proc. of the International Conference "33 Years of Groebner Bases", 1998, RISC, Austria, London Math. Society Lecture Note Series 251, pp. 535–545. Cambridge Univ. Press (1998))
13. Buchberger, B.: Gröbner bases: An algorithmic method in polynomial ideal theory. In: Bose, R. (ed.) Multidimensional Systems Theory, Mathematics and Its Applications, chap. 6, pp. 184–232. D. Reidel Publishing Company (1985)
14. Cox, D., Little, J., O'Shea, J.: Ideals, Varieties, and Algorithms An Introduction to Computational Algebraic Geometry and Commutative Algebra. Springer (1996)
15. Becker, T., Weispfenning, V.: Gröbner bases a computational approach to commutative algebra. In: Graduate Texts in Mathematics. Springer (1993)
16. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: towards a standard CP modelling language. In: Principles and Practice of Constraint Programming, pp. 529–543. Springer (2007)
17. Buchberger, B., Moeller, M.: The construction of multivariate polynomials with preassigned zeros. In: Computer Algebra, EUROCAM 82. LNCS, vol. 144, pp. 24–31 (1982)
18. Gault, R., Jeavons, P.: Implementing a test for tractability. J. Constraints **9**, 139–160 (2004)
19. Adams, W., Loustaunau, P.: An introduction to Gröbner bases. In: Graduate Studies in Mathematics. American Mathematical Society (1994)
20. Bayer, D., Mumford, D.: What can be computed in algebraic geometry? In: Eisenbud, D., Robbiano, L. (eds.) Computational Algebraic Geometry and Commutative Algebra, pp. 1–48. Cambridge University Press, Cambridge (1992)
21. Lazard, D.: Gröbner-bases, Gaussian elimination and resolution of systems of algebraic equations. In: Proceedings of the European Computer Algebra Conference on Computer Algebra, EUROCAL'83, pp. 146–156. Springer-Verlag, London (1983)
22. Cooper, M.: An optimal k-consistency algorithm. Artif. Intell. **41**, 89–95 (1989)
23. Freuder, E.: A sufficient condition for backtrack-bounded search. J. ACM **32**, 755–761 (1985)
24. Jeavons, P., Cooper, M.: Tractable constraints on ordered domains. Artif. Intell. **79**(2), 327–339 (1995)
25. Dechter, R.: Constraint Processing. Morgan Kaufmann, San Mateo (2003)
26. Dechter, R.: Bucket elimination: a unifying framework for reasoning. Artif. Intell. **113**(1–2), 41–85 (1999)
27. Jeavons, P., Petke, J.: Local consistency and SAT-solvers. J. Artif. Intell. Res. (JAIR) **43**, 329–351 (2012)
28. Faugére, J.C.: A new efficient algorithm for computing Gröbner Bases (F4). J. Pure Appl. Algebra **139**, 61–88 (1999)
29. Faugére, J.C.: FGb: a library for computing Gröbner Bases. In: Fukuda, K., Hoeven, J., Joswig, M., Takayama, N. (eds.) Mathematical Software - ICMS 2010. Lecture Notes in Computer Science, vol. 6327, pp. 84–87. Springer, Berlin Heidelberg (2010)