Applicable Algebra in Engineering, Communication and Computing
https://doi.org/10.1007/s00200-025-00684-8

ORIGINAL PAPER

A new iterative algorithm for comprehensive Gröbner systems

Anna Maria Bigatti[1] · Elisa Palezzato[2] · Michele Torielli[3]

Received: 21 April 2024 / Revised: 4 February 2025 / Accepted: 5 February 2025

Abstract
A comprehensive Gröbner system for a parametric ideal I in $K(A)[X]$ represents the collection of all Gröbner bases of the ideals I' in $K[X]$ obtained as the values of the parameters A vary in K. The recent algorithms for computing them consider the corresponding ideal J in $K[A, X]$, and are based on stability of Gröbner bases of ideals under specializations of the parameters A. Starting from a Gröbner basis of J, the computation splits recursively depending on the vanishing of the evaluation of some "coefficients" in $K[A]$. In this paper, taking inspiration from the algorithm described by Nabeshima, we create a new iterative algorithm to compute comprehensive Gröbner systems. We show how we keep track of the sub-cases to be considered, and how we avoid some redundant computation branches using "comparatively-cheap" ideal-membership tests, instead of radical-membership tests.

Keywords Comprehensive Gröbner systems · Gröbner basis · Algorithm · Radical membership

Mathematics Subject Classification 68W30 · 13P10

✉ Anna Maria Bigatti
a.m.bigatti@unige.it

✉ Elisa Palezzato
elisa.palezzato@gmail.com

✉ Michele Torielli
michele.torielli@nau.edu

[1] Dipartimento di Matematica, Università degli Studi di Genova, Via Dodecaneso 35, 16146 Genova, Italy

[2] Department of Mathematics, Hokkaido University, Kita 10, Nishi 8, Kita-Ku, Sapporo, Hokkaido 060-0810, Japan

[3] Department of Mathematics and Statistics, Northern Arizona University, 801 S. Osborne Dr., Flagstaff, AZ 86001, USA

Published online: 31 March 2025

1 Introduction

Let us start with a simple example to introduce the central object of this paper. Consider $\mathcal{C}_1$, a circle with center (0, 0) and radius 1, and $\mathcal{C}_2$, a circle with center $(c, 0)$ and radius $\sqrt{r}$. What is their intersection as the parameters c and r vary in $\mathbb{C}$? We can see that there are three cases: if $c = 0$, $r = 1$ the two circles coincide, if $c = 0$, $r \neq 1$ the two circles do not intersect, if $c \neq 0$ the two circles intersect in two points in $\mathbb{C}^2$ (may be one double point).

Algebraically speaking, the ideal $I = \langle x^2+y^2-1,\ (x-c)^2+y^2-r \rangle$, defined by the equations of $\mathcal{C}_1$ and $\mathcal{C}_2$, has a collection of different Gröbner bases depending on the values of the parameters c and r. This is described by a *comprehensive Gröbner system*, which is a set of pairs $\{(\mathcal{A}_1, G_1), \ldots, (\mathcal{A}_l, G_l)\}$, the Gröbner bases G_i under the conditions $\mathcal{A}_i$. For example, one such pair in our example is $\mathcal{A}_1 = \{(c, r) \in \mathbb{C}^2 \mid c \neq 0\}$ and its corresponding lex-Gröbner basis (lex with $x > y > c > r$)

$G_1 = \{\ x^2 + y^2 - 1,\ \ cx - \frac{1}{2}c^2 + \frac{1}{2}r - \frac{1}{2},\ \ xr - x - 2y^2c - \frac{1}{2}c^3 + \frac{1}{2}cr + \frac{3}{2}c,$
$c^2y^2 + \frac{1}{4}c^4 - \frac{1}{2}rc^2 - \frac{1}{2}c^2 + \frac{1}{4}r^2 - \frac{1}{2}r + \frac{1}{4}\ \}$

Comprehensive Gröbner systems for parametric ideals were introduced, constructed, and studied in 1992 by Weispfenning, see [13]. After Weispfenning's work, Kapur in [5] introduced an algorithm for parametric Gröbner bases. However, there was no big development on comprehensive Gröbner systems for around ten years. On the other hand, around 2003, big developments were made by Kapur-Sun-Wang [6], Montes [8], Nabeshima [9, 10], Sato and Suzuki [11, 12], and Weispfenning [14].

Some of these algorithms for computing comprehensive Gröbner systems are based on stability of Gröbner bases of ideals under specializations. Each algorithm has a different "stability condition" for monomial bases.

Notice that the study of comprehensive Gröbner systems using Nabeshima's approach has many applications. For example, we can classify the dimension of parametric ideals, study complex quantifier elimination (see [3]), and compute minimal and reduced comprehensive Gröbner systems without having to compute Gröbner basis in the parametric ring.

In this paper, taking inspiration from his work, we create a new iterative algorithm to compute comprehensive Gröbner systems.

In Sect. 2, we recall the definition of a comprehensive Gröbner system and we fix the notation and basic definitions needed for the rest of the paper. In Sect. 3, we describe the stability conditions used in previous papers to construct comprehensive Gröbner systems involving only computations in $K[A, X]$ instead of doing Gröbner basis computations in $K(A)[X]$. In Sect. 4, we describe our new iterative approach to compute comprehensive Gröbner system and we prove its correctness. In Sect. 5, we illustrate timings and statistics of our algorithm on known examples. In Sect. 6, we provide some ideas for further investigations and possible improvements.

2 Notation

Let K be a computable field, $\bar{K}$ its algebraic closure, X a set of N_X indeterminates, and A a set of N_A parameters. We work in the polynomial ring $K[A, X] = K[a_1, \dots, a_{N_A}, x_1, \dots, x_{N_X}]$.

We want to study what happens when we specialize the parameters A.

Definition 2.1 Fixed a point $P \in \bar{K}^{N_A}$, we indicate with $\mathrm{eval}_P(f)$ the evaluation homomorphism from $K[A]$ to $\bar{K}$, $f \mapsto f(P)$. Then we also indicate with eval_P its natural "coefficientwise" extension $K[A, X] \to \bar{K}[X]$, $f \mapsto f(P, X)$.

For example, given the point $P = (1, 3) \in \mathbb{Q}^2$ and $f = 3axy^2 - bxy^2 + ax^2 + b$ in $\mathbb{Q}[A, X]$, where $A = \{a, b\}$ and $X = \{x, y\}$, we have $\mathrm{eval}_P(f) = x^2 + 3 \in \mathbb{Q}[X]$.

Fixed a term-ordering τ on the power-products of $K[A, X]$, for a polynomial $f \in K[A, X]$ we denote by $\mathrm{LPP}(f) \in K[A, X]$ the τ-greatest power-product in the support of f, by $\mathrm{LC}(f) \in K$ its coefficient, and by $\mathrm{LM}(f) = \mathrm{LC}(f) \cdot \mathrm{LPP}(f)$.

Throughout this paper we will only consider *elimination orderings* for X, *i.e.* such that any power-product only in A is smaller than all power-products containing some $x_i \in X$. Such orderings reflect the interpretation of $K[A, X]$ as $K[A][X]$, with the polynomials in A seen as "parametric coefficients", and allow any choice of ordering on the restriction to $K[A]$ and $K[X]$.

Definition 2.2 Given on $K[A, X]$ an elimination ordering τ for X, we indicate with τ_X its restriction to the power-products only in X. Then we indicate with $\mathrm{LPP}_X(f) \in K[X]$ the τ_X-greatest power-product of f as a polynomial in $K[A][X]$, and with $\mathrm{LC}_X(f)$ its coefficient in $K[A]$.

Example 2.3 For $f = (a^2 - b)x^3y + 3abxy^2 \in \mathbb{Q}[a, b, x, y]$ equipped with any elimination ordering for $X = \{x, y\}$ such that $x^3y > xy^2$, e.g. $\tau = \mathrm{lex}$,

then we have $\mathrm{LPP}(f) = a^2x^3y$ with $\mathrm{LC}(f) = 1$, and $\mathrm{LPP}_X(f) = x^3y$ with $\mathrm{LC}_X(f) = a^2 - b$.

Throughout this paper, we indicate with gothic letters $\mathfrak{a}$, $\mathfrak{b}$, $\mathfrak{g}$ ideals in $K[A]$ (and their embedding in $K[A, X]$), and with I, J ideals in $K[A, X]$.

Moreover, we use the following notation $\mathbb{V}(\mathfrak{a}) = \{\alpha \in \bar{K}^{N_A} \mid f(\alpha) = 0, \forall f \in \mathfrak{a}\}$.

We'll see that many operations throughout the algorithms actually consist in comparing the "ideals of coefficients" $\mathfrak{a}$, $\mathfrak{b}$ in $K[A]$, for detecting whether $\mathbb{V}(\mathfrak{a}) \subseteq \mathbb{V}(\mathfrak{b})$ and then discard redundant computations branches (see Sect. 3.1).

Now we can formally define a *comprehensive Gröbner system*.

Definition 2.4 Given an ideal I in $K[A, X]$, a finite set of pairs, called **segments**, $\mathrm{CGS} = \{(\mathcal{A}_1, G_1), \dots, (\mathcal{A}_\ell, G_\ell)\}$ is a **comprehensive Gröbner system** for I if

- $\mathcal{A}_1, \dots, \mathcal{A}_\ell$ are *algebraically constructible* subsets of $\bar{K}^{N_A}$ (*i.e.* $\mathcal{A}_j = \mathbb{V}(\mathfrak{a}_j) \setminus \mathbb{V}(\mathfrak{b}_j)$ with $\mathfrak{a}_j, \mathfrak{b}_j$ ideals in $K[A]$) and $\cup \mathcal{A}_j = \bar{K}^{N_A}$

- $G_1, \ldots, G_\ell$ are finite subsets of $K[A, X]$
- for each *segment* $(\mathcal{A}_j, G_j)$ in CGS and for any point $P \in \mathcal{A}_j$ we have that $\mathrm{eval}_P(G_j)$ is a τ_X-Gröbner basis of the ideal $\mathrm{eval}_P(I)$ in $\bar{K}[X]$.

3 Stability and algorithms

In this section we recall how recent algorithms use stability conditions to construct comprehensive Gröbner systems involving only computations in $K[A, X]$ instead of having to track the arithmetic inside a Gröbner basis computation in $K(A)[X]$.

Suzuki-Sato in [12] observed that the stability theorem proved by Kalkbrener (for any ring homomorphism $\phi : K[A] \longrightarrow \bar{K}$), re-read in terms of eval_P, provided this new approach for computing comprehensive Gröbner systems.

Theorem 3.1 (Kalkbrener [4], Suzuki-Sato [12]) *Let I be an ideal in $K[A, X]$. Let τ be an X-elimination term-ordering on the power-products in $K[A, X]$ and G a τ-Gröbner basis of I.*

Fix a point $P \in \bar{K}^{N_A}$ such that $\mathrm{eval}_P(\mathrm{LC}_X(g)) \neq 0$ for each $g \in G \setminus (G \cap K[A])$. Then, $\mathrm{eval}_P\{G\}$ is a τ_X-Gröbner basis of $\mathrm{eval}_P(I)$.

We briefly illustrate with a small example how they applied this theorem for constructing a comprehensive Gröbner system.

Example 3.2 Consider again, from the Introduction, $I := \langle f, g \rangle \in \mathbb{C}[c, r, x, y]$, where $f := x^2 + y^2 - 1$ and $g := (x-c)^2 + y^2 - r$, and we fix the term-ordering lex with $x > y > c > r$, which is an elimination ordering for $\{x, y\}$. The first step of the algorithm consists in computing a lex-reduced Gröbner basis of I:

$G = \{\ \underline{x^2} + y^2 - 1,\quad \underline{cx} - \frac{1}{2}c^2 + \frac{1}{2}r - \frac{1}{2},$

$\underline{rx - x} - 2cy^2 - \frac{1}{2}c^3 + \frac{1}{2}rc + \frac{3}{2}c,\quad \underline{c^2y^2} + \frac{1}{4}c^4 - \frac{1}{2}rc^2 + \frac{1}{4}r^2 - \frac{1}{2}c^2 - \frac{1}{2}r + \frac{1}{4}\ \}.$

From Theorem 3.1 we know that G is a $\mathrm{lex}_{x,y}$-Gröbner basis of the specialization of I for any (c, r) in $\mathcal{A}_1 = \bar{\mathbb{Q}}^2 \setminus (\mathbb{V}(c) \cup \mathbb{V}(r-1) \cup \mathbb{V}(c^2))$. Then we recursively deal with the remaining zero cases $\mathbb{V}(c)$, $\mathbb{V}(r-1)$, $\mathbb{V}(c^2)$ by recursively running the algorithm on the ideals $I + \langle c \rangle$, $I + \langle r-1 \rangle$, $I + \langle c^2 \rangle$.

Remark 3.3 Notice that we can reduce the number of recursion steps by considering the radical of the ideals, in this example $\sqrt{\langle c^2 \rangle} = \langle c \rangle$ (see also Sect. 3.1). Unfortunately, computing the radical of an ideal is difficult to implement and possibly quite slow.

There is a delicate balance to keep in mind: reducing the number of cases to be considered, while controlling the cost for detecting them.

In this direction, further advance was achieved by proving that just a subset of the Gröbner basis in $K[A, X]$ needs to be considered. In [6] Kapur-Sun-Wang discovered a new stability condition, and in [10], Nabeshima improved it, further reducing redundant computation branches.

They are both centered on the following definition.

Definition 3.4 For a finite set $F \subseteq K[A, X]$ we define **MB(F)** to be the **Minimal Basis** of the monomial ideal $\langle \mathrm{LPP}_X(f) \mid f \in F \rangle \subset K[X]$ (e.g. in Example 3.2 we have $\mathrm{MB}(G) = \{x, y^2\}$).

Theorem 3.5 *Let I be an ideal in $K[A, X]$. Let τ be an X-elimination term-ordering on the power-products in $K[A, X]$ and G a τ-Gröbner basis of I.*

Let $\mathfrak{g} = G \cap K[A]$ and name $\{t_1, \ldots, t_s\}$ the power-products in $\mathrm{MB}(G \setminus \mathfrak{g})$.

- (Kapur-Sun-Wang [6]) For each t_i, select one polynomial $g_{t_i} \in G$ (not uniquely determined) such that $\mathrm{LPP}_X(g_{t_i}) = t_i$ Then, for all $P \in \mathbb{V}(\mathfrak{g}) \setminus (\mathbb{V}(\mathrm{LC}_X(g_1)) \cup \mathbb{V}(\mathrm{LC}_X(g_2)) \cup \cdots \cup \mathbb{V}(\mathrm{LC}_X(g_s)))$, $\mathrm{eval}_P(\{g_{t_1}, \ldots, g_{t_s}\})$ is a τ_X-Gröbner basis of $\mathrm{eval}_P(I)$ in $\bar{K}[X]$.
- (Nabeshima [10]) For each t_i, let $G_{t_i} = \{g \in G \mid \mathrm{LPP}_X(g) = t_i\}$. Then, for all $P \in \mathbb{V}(\mathfrak{g}) \setminus (\mathbb{V}(\mathrm{LC}_X(G_{t_1}) \cup \mathbb{V}(\mathrm{LC}_X(G_{t_2}) \cup \cdots \cup \mathbb{V}(\mathrm{LC}_X(G_{t_s}))$, $\mathrm{eval}_P(G_{t_1} \cup G_{t_2} \cup \cdots \cup G_{t_s})$ is a τ_X-Gröbner basis of $\mathrm{eval}_P(I)$ in $\bar{K}[X]$.

Example 3.6 Consider again the two circles from Example 3.2: we have $\mathrm{MB}(G) = \{x, y^2\}$. So by Theorem 3.5(Nabeshima) we have $\mathfrak{g} = \langle 0 \rangle$,

$G_x = \{\underline{cx} - \frac{1}{2}c^2 + \frac{1}{2}r - \frac{1}{2}, \quad \underline{rx} - 1x - 2cy^2 - \frac{1}{2}c^3 + \frac{1}{2}rc + \frac{3}{2}c\}$,

$G_{y^2} = \{\underline{c^2y^2} + \frac{1}{4}c^4 - \frac{1}{2}rc^2 + \frac{1}{4}r^2 - \frac{1}{2}c^2 - \frac{1}{2}r + \frac{1}{4}\}$.

Then, for all $P \in \bar{\mathbb{Q}} \setminus (\mathbb{V}(\langle c, r - 1 \rangle) \cup \mathbb{V}(\langle c^2 \rangle))$, $\mathrm{eval}_P(G_x \cup G_{y^2})$ is a $\mathrm{lex}_{\{x,y\}}$-Gröbner basis of $\mathrm{eval}_P(I)$ in $\bar{\mathbb{Q}}[x, y]$.

3.1 Avoiding redundancy

In Remark 3.3 we pointed out that we may compute *radicals* of the ideals $\mathfrak{a}$ in $K[A]$ because they describe the same set of points in $\bar{K}^{N_A}$, and, by doing so, we likely avoid some inconclusive intermediate step in the algorithm. The Gröbner basis computation of $I + \sqrt{\mathfrak{a}}$ should generally be easier (but not always so). The problem here is that computing $\sqrt{\mathfrak{a}}$ is hard to implement and may be quite slow to run, so the cost of such computation could be higher than the saving it produces.

In [10] redundancy is avoided by checking, before computing the Gröbner basis of $I + \mathfrak{a} \in K[A, X]$, whether or not $\mathbb{V}(\mathfrak{a})$ actually contains some new point not yet been considered in the previous recursive calls. The information of the processed sets of points is passed through the recursive calls in the form of one ideal: in fact,

note that $\mathbb{V}(\mathfrak{b}_1) \cup \mathbb{V}(\mathfrak{b}_2) = \mathbb{V}(\mathfrak{b}_1 \cdot \mathfrak{b}_2)$, so the union of all zero sets may be represented by just one ideal $\mathfrak{b}$. Then, by checking that $\mathbb{V}(\mathfrak{a}) \setminus \mathbb{V}(\mathfrak{b}) \neq \emptyset$, we guarantee that no point in $\bar{K}^{N_A}$ is considered twice. This condition, equivalent to $\mathfrak{a} \not\subseteq \sqrt{\mathfrak{b}}$, can be computed by *radical membership* without actually computing $\sqrt{\mathfrak{b}}$: e.g. $f \notin \sqrt{\mathfrak{b}}$ is equivalent to $\mathfrak{b} + \langle f-1 \rangle \neq \langle 1 \rangle$. Therefore, radical membership may be computed in any CAS and is generally cheaper than computing radicals. However, it is still quite expensive.

Here we propose a transversal approach: we convert the recursive structure of the algorithm into an iteration, collecting all the ideals yet to be considered into one set, VanishingToDo. The fact of having all ideals in one set, allows to apply radicals and/or radical membership if desired, but in addition provides further criteria based on simple ideal membership, which is faster to compute (see Remark 4.4).

4 The iterative algorithm

In general, it is always worth investigating whether a recursion could be translated into an iteration. Thinking this way may produce new insights and more efficient code. Instead of recursively passing the objects to be dealt with, we can collect them into one set (we call it VanishingToDo), and one by one we pick and process them.

This approach offers two general optimizations. One is that we can choose how to keep the set VanishingToDo "minimal" (where the meaning of "minimal" may be more than just removing duplicates), and the other is that we can choose which "object" to pick first.

Inspired by Nabeshima's [10] recursive algorithm, we designed this iterative algorithm.

We keep VanishingToDo "minimal" by discarding the ideals which include some other ideal within this set, and we prove that no segment is missed in the comprehensive Gröbner system, and that we can minimalize it efficiently (see Steps 2.5.2, 2.6.5, 2.6.6, and Theorem 4.3).

Algorithm 4.1 *CGS-Iter*

Algorithm 4.1. CGS-Iter
Input I ideal in $R = K[A, X]$ with an X-elimination ordering.

1 Initialization:
VanishingToDo:= $\{\langle 0\rangle\}$ (set of ideals in $K[A]$)
CGS := $\emptyset$ (set of pairs $(\mathcal{A}, G)$, with $\mathcal{A} \subset \bar{K}^{N_A}$ and $G \subset R$)
2 Main Loop: while VanishingToDo $\neq \emptyset$ do
2.1 $\mathfrak{a}$:= choose and remove one ideal from VanishingToDo
2.2 $J := I + \mathfrak{a}$
2.3 Compute G, a τ-Gröbner basis of J
2.4 $\mathfrak{g} := J \cap K[A]$ *(⟶ Note: $\mathfrak{g}$ is generated by $G \cap K[A]$, and $\mathfrak{a} \subseteq \mathfrak{g}$)*
2.5 if $\mathbb{V}(\mathfrak{a}) \backslash \mathbb{V}(\mathfrak{g}) \neq \emptyset$ then *(⟶ Note: $\mathfrak{a} \subsetneq \mathfrak{g}$)*
2.5.1 append the pair $(\mathbb{V}(\mathfrak{a}) \backslash \mathbb{V}(\mathfrak{g}), \{1\})$ to CGS
2.5.2 if $\mathfrak{g}$ contains no ideal in VanishingToDo then
append $\mathfrak{g}$ to VanishingToDo
2.6 else *(⟶ Note: $\mathbb{V}(\mathfrak{a}) = \mathbb{V}(\mathfrak{g})$)*
2.6.1 MB := $\mathrm{MB}(G \setminus \mathfrak{g}) \subseteq K[X]$
2.6.2 $G_{\mathrm{MB}} := \{g \in G \mid \mathrm{LPP}_X(g) \in \mathrm{MB}\} \subseteq G$
2.6.3 for each $t \in \mathrm{MB}$ let $\mathfrak{c}_t := \langle \mathrm{LC}_X(g) \mid g \in G_{\mathrm{MB}}, \mathrm{LPP}_X(g) = t\rangle \subseteq K[A]$
2.6.4 append the pair $(\mathbb{V}(\mathfrak{g}) \setminus (\bigcup_{t \in \mathrm{MB}} \mathbb{V}(\mathfrak{c}_t)),\ G_{\mathrm{MB}})$ to CGS
2.6.5 NewVanishing:= minimalized $\{\mathfrak{c}_t + \mathfrak{g} \mid t \in \mathrm{MB} \text{ and } \mathfrak{c}_t \not\subseteq \mathfrak{g}\}$
2.6.6 for each $\mathfrak{b}$ in NewVanishing do
if $\mathfrak{b}$ contains no ideal in VanishingToDo then
append $\mathfrak{b}$ to VanishingToDo
Output return CGS;

Theorem 4.2 *Let $R = K[A, X] = K[a_1, \ldots, a_{N_A}, x_1, \ldots, x_{N_X}]$ and τ and X-elimination ordering for the power-products in R. Let I be any ideal in R. Then*

1. *The algorithm CGS-Iter with input I terminates. Call* CGS_I *its output.*
2. *For each pair* $(\mathcal{A}, G)$ *in* CGS_I, $\mathrm{eval}_P(G)$ *is a* τ_X*-Gröbner basis for* $\mathrm{eval}_P(I)$ *for any point* $P \in \mathcal{A}$.
3. *Each point* $P \in \bar{K}^{N_A}$ *is in* (*at least*) *a set* $\mathcal{A}$ *in* CGS_I.

In summary, the output CGS_I is a τ-comprehensive Gröbner system for I.

Proof (1) Each ideal $\mathfrak{b}$ added to VanishingToDo contains $\mathfrak{g} = (I + \mathfrak{a}) \cap K[A]$, where $\mathfrak{a}$ is its "parent"

(Step 2.1). Let's see that $\mathfrak{b} \supsetneq \mathfrak{a}$. If added in Step 2.5.2 we have $\mathfrak{b} = \mathfrak{g} \supsetneq \mathfrak{a}$ because $\mathbb{V}(\mathfrak{a}) \backslash \mathbb{V}(\mathfrak{g}) \neq \emptyset$. If added in Step 2.6.6 we have $\mathfrak{b} \supsetneq \mathfrak{g} \supseteq \mathfrak{a}$, because $\mathfrak{b} = \mathfrak{c}_t + \mathfrak{g}$ where $\mathfrak{c}_t \not\subseteq \mathfrak{g}$. In conclusion, we build strictly increasing chains of ideals which are finite by the noetherianity of $K[A]$. Moreover, there are finitely many such chains because NewVanishing in Step 2.6.6 is finite, as MB is finite for the noetherianity of R.

(2) If $(\mathcal{A}, G)$ was added in Step 2.5.1, then $\mathcal{A} = \mathbb{V}(\mathfrak{a})\backslash\mathbb{V}(\mathfrak{g})$, thus $(\mathcal{A},\ \{1\})$ is a correct segment because $\mathfrak{g} \subseteq I + \mathfrak{a}$ therefore for any point $P \in \mathcal{A}$ the ideal $\mathrm{eval}_P(I + \mathfrak{a})$ contains a non-zero constant.

If $(\mathcal{A}, G)$ was added in Step 2.6.4 then $\mathcal{A} = \mathbb{V}(\mathfrak{a})\backslash(\bigcup_{t\in \mathrm{MB}} \mathbb{V}(\mathfrak{c}_t))$, thus $(\mathcal{A},\ G_{\mathrm{MB}})$ is a correct segment because each point in $\mathcal{A}$ satisfies the hypotheses of Theorem 3.5.Nabeshima for the ideal $I + \mathfrak{a}$.

(3) Let's consider first what happens in a single iteration of the main loop. Fix a point $P \in \mathbb{V}(\mathfrak{a})$ at Step 2.1. We claim that, before the next iteration, P is added to CGS or, to be treated in a later iteration, as a vanishing point in VanishingToDo.

If $\mathbb{V}(\mathfrak{a})\backslash\mathbb{V}(\mathfrak{g}) \neq \emptyset$ then

- if $P \in \mathbb{V}(\mathfrak{a})\backslash\mathbb{V}(\mathfrak{g})$ then it is contemplated in CGS (added in Step 2.5.1),
- else $P \in \mathbb{V}(\mathfrak{g})$. Then, $\mathfrak{g}$ is added to VanishingToDo, unless it contains some $\mathfrak{b}$ in VanishingToDo. In either case, P will be listed for consideration as point of $\mathbb{V}(\mathfrak{g})$ or of $\mathbb{V}(\mathfrak{b}) \subset \mathbb{V}(\mathfrak{g})$.

Otherwise, $\mathbb{V}(\mathfrak{a}) \subset \mathbb{V}(\mathfrak{g})$.

- If $P \in \mathbb{V}(\mathfrak{a})\backslash(\bigcup_{t\in \mathrm{MB}} \mathbb{V}(\mathfrak{c}_t))$ then P is contemplated in CGS (added in Step 2.6.4)
- else $P \in \mathbb{V}(\mathfrak{c}_t)$ for some $t \in \mathrm{MB}$. Then, $\mathfrak{c}_t$ is added to VanishingToDo, unless it contains some other $\mathfrak{c}_{t'}$ or $\mathfrak{b}$ in VanishingToDo. In either case, P will be listed for consideration as point of $\mathbb{V}(\mathfrak{c}_t)$ directly, or of $\mathbb{V}(\mathfrak{c}_{t'}) \subset \mathbb{V}(\mathfrak{c}_t)$, or of $\mathbb{V}(\mathfrak{b}) \subset \mathbb{V}(\mathfrak{c}_t)$, or of $\mathbb{V}(\mathfrak{b}) \subset \mathbb{V}(\mathfrak{c}_{t'}) \subset \mathbb{V}(\mathfrak{c}_t)$ (Steps 2.6.5 and 2.6.6). Notice that $P \in \mathbb{V}(\mathfrak{a}) \subset \mathbb{V}(\mathfrak{g})$, then $P \in \mathbb{V}(\mathfrak{c}_t) = \mathbb{V}(\mathfrak{c}_t + \mathfrak{g})$

This concludes the proof of the Claim.

Now, observe that the algorithm starts with $\langle 0\rangle$ in $K[A]$, so, at the termination of the algorithm, each point $P \in \bar{K}^{N_A}$ is in (at least) a set $\mathcal{A}$ in CGS_I. □

Although not optimal, we can prove that many redundant branches of computations are avoided by checking ideal membership. In fact, throughout the computation, the set of ideals VanishingToDo is naturally inclusion-minimalized and new additions to it are "inclusion-increasing". More precisely,

Theorem 4.3 *Throughout the computation, whenever an ideal* $\mathfrak{b}$ *is added to VanishingToDo*:

1. $\mathfrak{b} \not\supseteq \mathfrak{z}$ *for any* $\mathfrak{z}$ *in the current VanishingToDo*
2. $\mathfrak{b} \not\subseteq \mathfrak{z}$ *for any* $\mathfrak{z}$ *ever listed in VanishingToDo*
3. *VanishingToDo is minimalized* (*i.e. no ideal includes another ideal*)

Proof By induction. At the end of the first iteration VanishingToDo is either $\{\langle g\rangle\}$ (Step 2.5.2) or the minimalized set $\{\mathfrak{c}_t + \mathfrak{g} \mid t\in \mathrm{MB} \text{ and } \mathfrak{c}_t \not\subseteq \mathfrak{g}\}$ (Step 2.6.5), and all 3 statements are trivially satisfied.

Let's consider an iteration starting at Step 2.1, picking $\mathfrak{a}$ from VanishingToDo; VanishingToDo is minimalized by induction, so $\mathfrak{a} \not\subseteq \mathfrak{z}$ for any $\mathfrak{z} \in$ VanishingToDo. We have $\mathfrak{a} \subseteq \mathfrak{g}$ (Step 2.4).

1. $\mathfrak{b} \not\supseteq \mathfrak{z}$ is explicitly checked in either cases: $\mathfrak{b} = \mathfrak{g}$ (Step 2.5.2) or $\mathfrak{b} \in$ NewVanishing (Step 2.6.6).
2. If we add $\mathfrak{b} = \mathfrak{g}$ in Step 2.5.2, then $\mathfrak{a} \subsetneq \mathfrak{g}$. If we add $\mathfrak{b}$ from NewVanishing in Step 2.6.6, then $\mathfrak{a} \subseteq \mathfrak{g} \subsetneq \mathfrak{b}$. In either cases, $\mathfrak{b} \not\subseteq \mathfrak{z}$ for any ideal $\mathfrak{z}$ ever listed in VanishingToDo, because so was $\mathfrak{a}$ (by inductive hypothesis), and because $\mathfrak{a} \subsetneq \mathfrak{b}$.
3. In particular, by (1) and (2), it follows that the current VanishingToDo is minimalized.

□

Remark 4.4 We just proved that many redundant computations are avoided thanks to the minimalizations (with respect to ideal inclusion) in Steps 2.6.5 and 2.6.6, and discarding ideals in Step 2.5.2.

Compared with the check $\mathbb{V}(\mathfrak{a})\backslash\mathbb{V}(\mathfrak{b}) \neq \emptyset$ described by Nabeshima in [10] (within these algorithms equivalent to the condition $\sqrt{\mathfrak{a}} \neq \sqrt{\mathfrak{b}}$), our approach misses to detect some redundant branches, but it is computationally cheaper because it is based on *ideal membership*, which can use many times the same pre-computed Gröbner bases of the ideals, instead of computing the radicals, or the *radical membership* (which needs a specific Gröbner basis for each pair $\mathfrak{a}, \mathfrak{b}$ – see Sect. 3.1).

Notice that we kept the check $\mathbb{V}(\mathfrak{a})\backslash\mathbb{V}(\mathfrak{g}) \neq \emptyset$ in Step 2.5. Choosing when and where to use this more effective+expensive check, is indeed a delicate balance.

Remark 4.5 In Step 2.5.2 we just append the ideal $\mathfrak{g}$ to VanishingToDo (if needed) and then pass to the next iteration, instead of going on and consider the ideals given by the LC_X in G (as in Step 2.6). By so doing, we "waste" the information included in the Gröbner basis G we just computed, but on the other hand, the fact that $\mathfrak{g}$ is quite different form its "ancestor" $\mathfrak{a}$ (it has a different vanishing set) make us believe that we'd better reconsider $\mathfrak{g}$ in the "whole picture", as a new element in the VanishingToDo list (if needed). Moreover, before checking and appending it, we make its generators square-free, so that, when it is processed in a following iteration (Step 2.1), it is "closer" to its radical and may reduce the number of the subsequent computations.

5 Timings

In this table we compare our iterative algorithm, "iter", with our best implementations of Nabeshima's algorithm, "N" and "N*", in CoCoA−5.4.1v, running on a MacBookPro (i7, 2.3GHz).

In "N*" we disabled Nabeshima's "RoughCheck", because this operation is costly and may be intended as a post-processing applicable at the end of any algorithm for

removing those segments whose support set is empty. Thus "N*" takes less time and returns a higher number of segments.

We used the examples listed in the papers [10, 6] and [7], and we encountered some discrepancies, so we indicate both names. Moreover, for some examples, the time of the very first GB in CoCoA is higher than the time obtained by the other authors for the whole computation, thus we are not quite sure we are using the same conditions/definitions. In particular, CoCoA could not compute the GBasis of Nabeshima's example S2 with $a > b > c > d$, so we changed the example into $b > c > d > a$.

We tested the 3 algorithms on these examples:

- 3 in SuzukiSato
- 4 in SuzukiSato
- F8 in Nabeshima
- E5 in Nabeshima
- S2 in KapurSunWang (similar to S4 in Nabeshima)
- S2 in Nabeshima – modified with indeterminates: b,c,d,a
- S3 in Nabeshima, same as S1 in KapurSunWang
- S4 in Nabeshima
- S3 in KapurSunWang (similar to S5 in Nabeshima)
- S5 in Nabeshima
- P3P in Nabeshima and in KapurSunWang

At the following link https://sites.google.com/view/bigatti-data the interested reader will find our CoCoA-5 package, CGS.cpkg5 (which is part of the forthcoming release CoCoA−5.4.2), together with the file containing all examples (which we used to generate the table of timings).

	iter	#	time	N*	#	time	N	#	time
SuzukiSato 3		30	0.4s		28	0.2s		19	0.3s
SuzukiSato 4		8	0.1s		9	0.1s		8	0.1s
Nabeshima F8		24	0.3s		23	0.3s		18	0.4s
Nabeshima E5		5	0.1s		8	0.2s		8	0.2s
KapurSunWang S2		4	0.2s		4	0.3s		4	0.5s
Nabeshima S2 modified		7	7.4s		8	7.2s		8	7.4s
Nabeshima S3		11	0.8s		14	3.5s		10	5.0s
Nabeshima S4		18	8.9s		21	28.6s		15	32.3s
KapurSunWang S3		63	2.0s		31	19.2s		18	31.1s
Nabeshima S5		79	2.5s		34	18.8s		20	31.2s
Nabeshima P3P		22	5.9s		20	13.9s		19	17.3s

Looking at the most interesting examples in the second half of the table, we see that *iter* is generally quite faster than *N** (*i.e. N* without the costly final minimalization of the segments), and sometimes also produces fewer segments.

To further investigate the different routes followed by the two algorithms, we collect in a more detailed table the the number of calls and the time spent for some crucial operations:

GB in $\boldsymbol{K[A,X]}$ the Gröbner basis computations of $I + \mathfrak{a} \in K[A,X]$ with an elimination ordering for X. This may be quite time consuming, especially for the very first computation: the Gröbner basis of the input ideal I.

GB in $K[A]$ the Gröbner basis computations of $\mathfrak{a} \in K[A]$: this is information is extracted from its actual application within "check $\mathfrak{a} \subseteq \mathfrak{b}$" and "check $\mathbb{V}(\mathfrak{a})\backslash\mathbb{V}(\mathfrak{b})=\emptyset$ ".

check $\mathfrak{a} \subseteq \mathfrak{b}$ only in our iterative algorithm: ideal membership $f \in \mathfrak{b}$, using the Gröbner basis of $\mathfrak{b}$

check $\mathbb{V}(\mathfrak{a})\backslash\mathbb{V}(\mathfrak{b})=\emptyset$ ("Checking consistency" in [10] radical membership $f \in \sqrt{\mathfrak{b}}$, checking first if $f \in \mathfrak{b}$ (using "GB in $K[A]$"), and then using the Gröbner basis of $\mathfrak{b} + \langle 1 - t{\cdot}f \rangle$.

product only in Nabeshima's algorithm: the product of the ideals, N, passed recursively as second argument

MB the computations of the minimal basis of the monomial ideal $\langle \mathrm{LPP}_X(f) \mid f \in GB \rangle \subset K[X]$ (Definition 3.4)

sqfr the computations for making square-free the generators of the ideals in $K[A]$ (Steps 2.5.2 and 2.6.5, and Remark 4.5) without computing the actual radical of the ideal.

last column final number of segments in the resulting CGS, and the total time for its computation.

In this table we observe that most of the time is spent in computing the GB in $K[A, X]$ and the operations involved in removing redundant computational branches (GB in $K[A]$, product of ideals, the checks themselves). The complete algorithm "N" also performs "RoughCheck" which removes some useless segment from the final output.

	GB in $K[A,X]$		GB in $K[A]$		check $\mathfrak{a} \subseteq \mathfrak{b}$		check $=\emptyset$ $\mathbb{V}(\mathfrak{a})\backslash\mathbb{V}(\mathfrak{b})$		product		MB		sqfr		#	time
SuzukiSato 3																
iter	30	0.1s	175	0.0s	419	0.0s	30	0.0s	-	-	29	0.0s	268	0.0s	30	0.4s
N*	27	0.1s	161	0.0s	-	-	187	0.0s	159	0.0s	26	0.0s	242	0.0s	28	0.2s
N	27	0.1s	172	0.0s	-	-	299	0.1s	271	0.0s	26	0.0s	242	0.0s	19	0.3s
SuzukiSato 4																
iter	8	0.0s	40	0.0s	54	0.0s	8	0.0s	-	-	8	0.0s	83	0.0s	8	0.1s
N*	9	0.0s	42	0.0s	-	-	50	0.0s	40	0.0s	9	0.0s	91	0.0s	9	0.1s
N	9	0.0s	51	0.0s	-	-	80	0.0s	70	0.0s	9	0.0s	91	0.0s	8	0.1s
Nabeshima F8																
iter	24	0.1s	109	0.0s	306	0.0s	24	0.0s	-	-	24	0.0s	215	0.0s	24	0.3s
N*	23	0.1s	107	0.0s	-	-	129	0.1s	105	0.0s	23	0.0s	207	0.0s	23	0.3s
N	23	0.1s	115	0.0s	-	-	195	0.1s	171	0.0s	23	0.0s	207	0.0s	18	0.4s
Nabeshima E5																
iter	5	0.0s	10	0.0s	8	0.0s	5	0.0s	-	-	4	0.0s	48	0.0s	5	0.1s
N*	4	0.0s	10	0.0s	-	-	13	0.1s	8	0.0s	4	0.0s	42	0.0s	8	0.2s
N	4	0.0s	12	0.0s	-	-	17	0.1s	12	0.0s	4	0.0s	42	0.0s	8	0.2s
(similar to Nabeshima S4) S2 KapurSunWang																
iter	4	0.0s	7	0.0s	2	0.0s	4	0.1s	-	-	3	0.0s	58	0.0s	4	0.2s
N*	3	0.0s	7	0.0s	-	-	9	0.2s	5	0.0s	3	0.0s	57	0.0s	4	0.3s
N	3	0.0s	8	0.0s	-	-	11	0.4s	7	0.0s	3	0.0s	57	0.0s	4	0.5s
Nabeshima S2 – modified –> b,c,d,a																
iter	7	7.2s	20	0.0s	22	0.0s	7	0.0s	-	-	7	0.0s	129	0.0s	7	7.4s
N*	8	6.9s	24	0.0s	-	-	31	0.2s	22	0.0s	8	0.0s	145	0.0s	8	7.2s
N	8	6.8s	26	0.0s	-	-	45	0.5s	36	0.0s	8	0.0s	145	0.0s	8	7.4s
Nabeshima S3 = S1 KapurSunWang																
iter	11	0.5s	34	0.1s	64	0.0s	11	0.0s	-	-	11	0.0s	186	0.1s	11	0.8s
N*	14	0.4s	45	0.1s	-	-	58	1.7s	43	1.0s	14	0.0s	177	0.1s	14	3.5s
N	14	0.4s	48	0.1s	-	-	84	2.5s	69	1.6s	14	0.0s	177	0.1s	10	5.0s
Nabeshima S4																
iter	18	8.6s	51	0.1s	83	0.0s	18	0.0s	-	-	18	0.0s	236	0.1s	18	8.9s
N*	21	9.0s	62	0.1s	-	-	82	7.3s	60	10.6s	21	0.0s	323	0.1s	21	28.6s
N	21	9.0s	65	0.1s	-	-	119	9.5s	97	11.9s	21	0.0s	323	0.1s	15	32.3s
(similar to Nabeshima S5) S3 KapurSunWang																
iter	63	0.8s	277	0.2s	831	0.0s	63	0.1s	-	-	60	0.1s	975	0.2s	63	2.0s
N*	31	0.4s	133	0.1s	-	-	163	7.8s	131	9.0s	28	0.0s	425	0.1s	31	19.2s
N	31	0.4s	138	0.1s	-	-	226	19.5s	194	9.2s	28	0.0s	425	0.1s	18	31.1s
Nabeshima S5																
iter	79	0.8s	416	0.2s	1315	0.0s	79	0.1s	-	-	79	0.1s	1202	0.2s	79	2.5s
N*	34	0.4s	174	0.1s	-	-	207	7.6s	172	8.9s	34	0.0s	485	0.1s	34	18.8s
N	34	0.4s	179	0.1s	-	-	299	19.4s	264	9.4s	34	0.0s	485	0.1s	20	31.2s
Nabeshima P3P = P3P KapurSunWang																
iter	22	1.1s	60	0.2s	94	0.0s	22	4.0s	-	-	21	0.0s	510	0.3s	22	5.9s
N*	18	1.0s	49	0.2s	-	-	66	4.7s	47	6.7s	16	0.0s	329	0.2s	20	13.9s
N	18	1.1s	52	0.2s	-	-	95	7.4s	76	7.2s	16	0.0s	329	0.2s	19	17.3s

6 Conclusion and future work

We believe that tackling the problem with an iterative approach allows to have a better perspective on the computation: having the cases yet to be considered altogether in a single list makes it possible to compare them with the looser but faster ideal membership, instead of keeping track of them through the recursion with product of ideals and the costlier radical membership. Moreover, we expect that further new strategies may be developed for optimizing the computations.

In particular, we think that there are two main areas that can be studied further:

- Strategies for picking the next ideal $\mathfrak{a} \subseteq K[A]$ to be considered (Step 2.1, function `ChooseVanishing`). Currently we randomly pick one out of those with the maximum dimension.
- Study possible criteria to minimize VanishingToDo, in order to reduce even further the cases that we analyze. Currently we keep in the list the minimal ones (Steps 2.5.2 and 2.6.6), so that we have "more zeroes", and we make their generators square-free. But ideally with should keep the radicals of these ideals, or just keep the larger when two ideals sharing the same radical. Thus, finding a cheap check to detect "more zeroes or closer to the radical" should be a further improvement.

Acknowledgements The authors would like to thank Katsusuke Nabeshima, Yosuke Sato and Miwa Taniwaki. This research was partly supported by the "National Group for Algebraic and Geometric Structures, and their Applications" (GNSAGA-INdAM).

Funding Open access funding provided by Università degli Studi di Genova within the CRUI-CARE Agreement.

References

1. Abbott, J., Bigatti, A. M.: CoCoALib: a C++ library for doing computations in commutative algebra, Available at https://github.com/cocoa-official/CoCoALib
2. Abbott, J., Bigatti, A.M., Robbiano, L.: CoCoA: a system for doing computations in commutative algebra, Available at https://sites.google.com/view/cocoa-cocoalib/download-cocoa5
3. Fukasaku, R., Inoue, S., Sato, Y.: On QE algorithms over an algebraically closed field based on comprehensive Gröbner systems. Math. Comput. Sci. **9**, 267–281 (2015)
4. Kalkbrener, M.: On the stability of Gröbner bases under specializations. J. Symb. Comput. **24**, 51–58 (1997)

5. Kapur, D.: An approach for solving systems of parametric polynomial equations. In: Saraswat, V., Hentenryck, P. (eds.) Principles and Practice of Constraint Programming, pp. 217–244. MIT Press (1995)
6. Kapur, D., Sun, Y., Wang, D.: A new algorithm for computing comprehensive Gröbner systems. In: Watt, S. (ed.) International Symposium on Symbolic and Algebraic Computation, pp. 29–36. ACM Press (2010)
7. Kapur, D., Sun, Y., Wang, D.: An efficient algorithm for computing a comprehensive Gröbner system of a parametric polynomial system. J. Symb. Comput. **49**, 27–44 (2013)
8. Montes, A.: A new algorithm for discussing Gröbner basis with parameters. J. Symb. Comput. **33**(1–2), 183–208 (2002)
9. Nabeshima, K.: A speed-up of the algorithm for computing comprehensive Grö-bner systems. In: Brown, C. (ed.) International Symposium on Symbolic and Algebraic Computation, pp. 299–306. ACM Press (2007)
10. Nabeshima, K.: Stability conditions of monomial bases and comprehensive Gröbner systems. Lect. Notes Comput. Sci. **7442**, 248–259 (2012)
11. Suzuki, A., Sato, Y.: An alternative approach to comprehensive Gröbner bases. J. Symb. Comput. **36**(3–4), 649–667 (2003)
12. Suzuki, A., Sato, Y.: A simple algorithm to compute comprehensive Gröbner bases using Gröbner bases. In: Dumas, J.-G. (ed.) International Symposium on Symbolic and Algebraic Computation, pp. 326–331. ACM Press (2006)
13. Weispfenning, V.: Comprehensive Gröbner bases. J. Symb. Comput. **14**(1), 1–29 (1992)
14. Weispfenning, V.: Canonical comprehensive Gröbner bases. J. Symb. Comput. **36**(3–4), 669–683 (2003)